# Analysing Uncertainty in Release Planning: A Method and Experiment for Fixed-Date Release Cycles

OLAWOLE ONI and EMMANUEL LETIER, Department of Computer Science, University College London, United Kingdom

Release planning —deciding what features to implement in upcoming releases of a software system— is a critical activity in iterative software development. Many release planning methods exist but most ignore the inevitable uncertainty in estimating software development effort and business value. The paper's objective is to study whether analysing uncertainty during release planning generates better release plans than if uncertainty is ignored. To study this question, we have developed a novel release planning method under uncertainty, called BEARS, that models uncertainty using Bayesian probability distributions and recommends release plans that maximise expected net present value and expected punctuality. We then compare release plans recommended by BEARS to those recommended by methods that ignore uncertainty on 32 release planning problems. The experiment shows that BEARS recommends release plans with higher expected net present value and expected punctuality than methods that ignore uncertainty, thereby indicating the harmful effects of ignoring uncertainty during release planning. These results highlight the importance of eliciting and analysing uncertainty in software effort and value estimations and call for increased research in these areas.

CCS Concepts: • **Software and its engineering → Software development process management**; **Risk management**.

Additional Key Words and Phrases: release planning, Bayesian analysis, decision analysis, value-based software engineering, search-based software engineering, software economics

**ACM Reference Format:**
Olawole Oni and Emmanuel Letier. 2021. Analysing Uncertainty in Release Planning: A Method and Experiment for Fixed-Date Release Cycles. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2021), 40 pages. https://doi.org/10.1145/3490487

## 1 INTRODUCTION

Iterative software development processes are widely recommended when requirements are uncertain and a project is subject to significant risks [12, 37]. Instead of delivering all features at once, the software is delivered over multiple releases where each release adds or modifies features based on lessons learnt from earlier releases. By feature, here, we mean both functional features and quality improvements (e.g. better performance). Since not all features are delivered at once, an essential activity of iterative development consists in prioritizing what features to deliver in upcoming releases. This activity, known as *release planning*, is critical to deliver early business value, to manage stakeholders' expectations, to control risks, and to organise software development activities [61].

Many release planning methods exist (see [5, 68] for surveys). These methods involve evaluating candidate features against a set of criteria relevant to the project stakeholders. Commonly used

Authors' address: Olawole Oni, olawole.oni.14@ucl.ac.uk; Emmanuel Letier, e.letier@ucl.ac.uk, Department of Computer Science, University College London, London, United Kingdom.

1049-331X/2021/1-ART1 $15.00
https://doi.org/10.1145/3490487

criteria include development effort measured in person-days or story points, and business value measured in abstract value points (e.g. a number from 0 to 9) or in monetary units (e.g. in $). Different methods propose different sets of criteria, different ways to elicit scores from stakeholders, and different mathematical models that combine the elicited scores into some overall metrics that inform the release planning decisions.

Empirical studies in industrial contexts have shown the benefits of systematic release planning methods over unstructured, ad-hoc release planning [4, 45, 51]. By introducing structure to the decision process, these methods mitigate common difficulties of release planning such as the lack of clarity about objectives and candidate features, the late identification of feature dependencies, the difficulty of engaging key stakeholders in the decision process, and the uncontrolled interference of power, politics and self-serving interests. The studies also showed that introducing a structured process reduces the time and effort involved in making decisions.

So far, however, no research has compared structured release planning methods against each other. Would different methods applied in the same context recommend the same release plans? If not, do some methods recommend better release plans than others? The paper's objective is to study whether methods that analyse uncertainty would recommend better release plans than methods that ignore uncertainty.

Most release planning methods ignore the inevitable uncertainty of software development effort and value. They evaluate candidate release plans as if all features will be delivered on time according to plan. In reality, some features will take longer to develop than predicted and, once released, may deliver more or less business value than expected. Ignoring such uncertainty can lead to inaccurate, overoptimistic, and inconsistent evaluation of candidate release plans, which in turn can lead to misinformed and possibly inadequate release planning decisions.

A few methods have been proposed to analyse uncertainty during release planning (e.g. [10, 46, 49, 58, 63]). However, no evidence exists that release plans recommended by these methods are better, or even just different, than release plans recommended by methods that ignore uncertainty. Without such evidence, the added complexity of analysing uncertainty cannot be justified and methods that analyse uncertainty are unlikely to be adopted.

Another limitation of previous release planning methods under uncertainty is their implicit assumption that all features planned for a release will be delivered in that release. If development takes longer than anticipated, they assume that the release date will be postponed until all features planned for the release are ready to be delivered. The methods analyse uncertainty about release dates and costs under that assumption. We call this assumption the *fixed-scope assumption* because it treats release plans as fixed-scope contracts. This assumption rarely holds in practice. Surveys indicate that most organisations use a *fixed-date release cycle* where future releases are scheduled to occur at fixed dates separated by regular time intervals (e.g. every 3 months) [45]. In this context, if development tasks take longer than anticipated, features that are not ready for their planned release date will be postponed to future releases or possibly cancelled. Existing release planning methods that assume fixed-scope releases are inadequate in this context because they are designed to analyse uncertainty about release dates (which is irrelevant in this context) and are unable to analyse uncertainty about the releases' content.

To study release planning in a context relevant to industrial practice, we have developed a novel release planning method, called BEARS, that analyses uncertainty in the context of fixed-date release cycles. BEARS is an acronym for Bayesian Economic Analysis of Release Scenarios. It is also a nod to "Waltzing with Bears", the title of a seminal book on managing uncertainty in software projects [18]. BEARS is a Bayesian approach in the sense that it uses Bayesian probability (i.e. probability representing someone's degree of belief about an uncertain proposition [56]) to model the release planners' uncertainty about the development effort and business value of candidate features. In this paper, we rely on the elicitation of such uncertainty from human experts rather than inferring them from a range of inputs using Bayesian Networks (a topic for future work). BEARS is an economic approach in the sense that it compares release plans' value using the financial concept of net present value. BEARS is supported by a tool that takes as input Bayesian probabilities of the effort and value of individual features and generates as output a set of Pareto-optimal release plans that maximize expected net present value (a common financial metric for comparing cash flows) and expected punctuality (the expected percentage of features delivered on time). Release planners (the persons responsible for release planning ) will then select their preferred release plan from the recommended Pareto-optimal set.

To study the impact of analysing uncertainty on release planning decisions, we have applied BEARS to 32 release planning problems and compared the release plans recommended by BEARS to release plans recommended by methods that ignore uncertainty. The experiment shows that methods that ignore uncertainty recommend release plans that have lower expected net present value and lower expected punctuality than those recommended by BEARS. In our experiment, in 97% of cases, *all* release plans shortlisted by methods that ignore uncertainty have strictly lower expected net present value and expected punctuality than the release plans shortlisted by BEARS. On average, BEARS increases the hypervolume of the shortlisted release plans by 18% — this roughly means that release planners have a choice of release plans that is 18% superior in terms of expected net present value and expected punctuality (the hypervolume metric will be defined more precisely later). These numbers are specific to our 32 release planning problems and the conditions of our controlled experiment. Since our controlled experiment is designed to *minimize* differences between the application of BEARS and methods that ignore uncertainty, the differences in practice may be much larger. More importantly, BEARS enables informed discussions about tradeoffs between expected value and expected punctuality that are impossible when uncertainty is ignored. The BEARS tool and all data needed to replicate our experiments are available from the paper's repository [1].

## 2 ILLUSTRATIVE EXAMPLE

Throughout the paper, we illustrate release planning with an example based on a real local government project. The project aims to develop online services for local residents and businesses for various tasks such as paying local taxes, reporting missed rubbish collection, managing parking permits, and submitting and responding to planning applications. Many residents and businesses would prefer to perform such transactions online instead of over the phone or by visiting offices when possible. For the local government, each online interaction is much cheaper than the equivalent interaction over the phone or in person. The local government has identified 15 individual services that could be delivered online but its Information System group has limited resources and is only able to develop a few features supporting these services every quarter. Different stakeholders disagree about which features should be developed first. The project manager and software development team need to prepare a release plan that will organise the development work so that it provides the most value to the local government and its residents. As for most release planning problems, substantial uncertainty exist about the value and development time of each online feature.

## 3 RELEASE PLANNING: BACKGROUND & STATE-OF-THE-ART

This section provides a guided tour of important concepts and methods of software release planning. Unlike previous surveys [5, 68], it emphasises the mathematical optimisation models at the heart of release planning methods and discusses the assumptions and limitations of these models. The

[1]https://github.com/olawole/BEARS

models described here provide foundations for the new model in Section 4. They will also be used in the experiment in Section 5.

Release planning involves planning several months ahead and is performed on coarse-grained work items such as features and epics rather than fine-grained work items such as user stories and small development tasks [38, 61]. In agile development, release planning is followed by more frequent sprint planning activities [38]. A sprint is a smaller development period (e.g. 2 weeks) within a longer release period (e.g. 3 months). Sprint planning focus on the next sprint only. Release planning in contrast usually involves planning multiple releases. In the academic literature, the activity studied in this paper is sometimes referred to as *strategic* release planning and sprint planning as *operational* release planning [2, 68]. The first is concerned with assigning coarse-grained work items to future releases, the second with assigning fine-grained work items to developers for the next short development period.

## 3.1 Product Backlog and Release Plans

Release planning takes as input a backlog of work items to be scheduled for future releases and generates as output a release plan specifying what items are scheduled for what future releases.
**Definition (Product Backlog):** A *product backlog* is

- a set $WI$ of work items that are candidates for future releases; and
- a relation $\leftarrow \subseteq WI \times WI$ defining a precedence relation between work items, i.e. $w_i \leftarrow w_j$ means that $w_i$ must be delivered before or at the same time as $w_j$.

In some release planning methods, backlog items are called requirements [9, 26], in others they are called features [19, 38, 61]. In this paper, we will follow the terminology of the Incremental Funding Method where work items are either features or architectural elements [19]. Features are functional or quality improvement that deliver value to stakeholders; *architectural elements* are software elements that do not in themselves deliver value to stakeholders but are prerequisites to the development of features.

In addition to the precedence relation, the product backlog may include other types of dependencies between work items such as coupling, exclusion, and weak precedence [61, 72]. We will not use these additional dependencies in this paper; all techniques described in the paper can however be extended to handle these additional dependency types (the required changes will be discussed in Section 4.3).

As an example, Figure 1 shows a fragment of the product backlog for our local government project. The full backlog, available in BEARS's online repository, is composed of 15 features, 5 architectural elements, and 23 precedence links.

The main output of release planning is a release plan.

**Definition (Release Plan):** Given a product backlog $< WI, \leftarrow >$, a release plan is a partial function $p : WI \rightarrow [1..H]$ that maps work items to future releases. The number $H \in \mathbb{N}^+$ of releases in the plan is called the *planning horizon*. For a work item $w$, $p(w) = i$ means that $w$ is planned to be delivered in the $i^{th}$ next release. A release plan is a partial function because not all work items in the backlog need to be planned for some future release. Release plans must satisfy the precedence constraints: if $w_i \leftarrow w_j$ then $p(w_i) \leq p(w_j)$.

## 3.2 Managing Capacity Constraints

Release planning must ensure that for each release period the required development effort does not exceed the development team's capacity. To model this constraint, the product backlog is extended with the following information:

- $\mathit{effort}(w) \in \mathbb{N}^+$ denoting the estimated effort required to deliver each $w \in WI$;

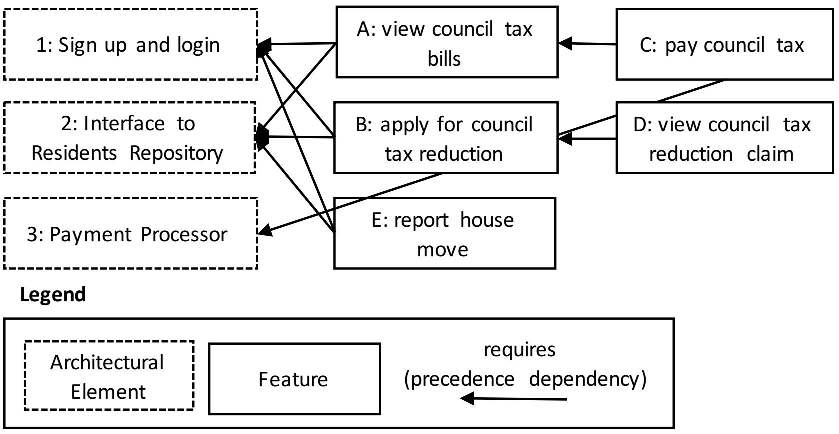

1: Sign up and login
2: Interface to Residents Repository
3: Payment Processor
A: view council tax bills
B: apply for council tax reduction
C: pay council tax
D: view council tax reduction claim
E: report house move
Legend
Architectural Element
Feature
requires (precedence dependency)

Fig. 1. Part of the product backlog for the local government project

- *capacity*$(i) \in \mathbb{N}^+$ denoting the estimated team capacity in each release period $i \in [1..H]$.

Effort and capacity estimates are provided by the development team and release planners. Techniques to support such estimations include planning poker [15] and formal estimation models [33]. Effort and capacity must be expressed in the same unit, for example in person-days or story points. Story points are popular in lean and agile software development. They denote abstract scores on a ratio scale (e.g. a score between 0 and 100) that denote the *relative* effort needed to deliver a work item in comparison to other work items (e.g. an item worth 50 story points is expected to require 5 times the effort of an item worth 10 story points). Effort estimation in person-days denote, as the name implies, the estimated number of person-days required to deliver an item.

Release plans must satisfy the constraint that for each release period, the development effort does not exceed the team capacity, i.e. for all $i \in [1..H]$:

$$\sum_{\{w \in WI \mid p(w)=i\}} \mathit{effort}(w) \leq \mathit{capacity}(i) \tag{1}$$

Constraints on other resources such as personnel, equipments, and capital can be modelled in the same way by specifying the resource requirements for each work item and the resource capacity during each period [61].

A well-known problem in managing capacity constraints is the difficulty of predicting development effort: effort estimates are often optimistic and ignore the inevitable uncertainty of software development tasks [32]. Ignoring such uncertainty, or at least not describing it explicitly, prevents release planners from analysing the likelihood and impacts of late deliveries.

One proposed approach to deal with this problem consists in assigning a risk level to each work item, e.g. on a scale from 0 to 5, and to tolerate a maximal level of risk in each iteration, e.g. at most 10 risk units per iteration [61]. In other words, risk is treated as a limited resource and managed using a capacity constraint. This approach however provides only a limited form of uncertainty analysis: the likelihood and impacts of late deliveries remain ambiguous and unquantified.

Other release planning methods address this problem by estimating effort as probability distributions. For example, effort uncertainty can be modelled using triangular distributions defined by the most optimistic, most likely and most pessimistic effort estimates [41, 42, 46, 63]. It has also been modelled using lognormal distributions [49]. Given these probability distributions, the above methods aim to maximise the probability that the next release is delivered within budget [41, 42] or by its planned date [46, 49]. These methods, however, are limited to analysing a single next release (i.e. their planning horizon H is limited to 1). Another method, dealing with multiple releases, aims to maximize the probability that all planned releases are delivered on time [63]. However, all these methods assume that a new release is delivered only when *all* features planned for that release have been completed. As mentioned in introduction, this fixed-scope assumption is unrealistic and inadequate for reasoning about uncertainty in the context of fixed-date release cycles most commonly used in industry.

## 3.3 Optimizing Value Points

Software systems are built to deliver value to their stakeholders [11]. To help release planners identify which release plans are likely to deliver most value, many release planning methods rely on assigning value points to release plans. In these methods, value points are abstract scores denoting *relative* value on a ratio scale (e.g. between 0 and 100); they do not correspond to concrete observable quantities (e.g. financial gain, number of users of the online services).

The product backlog is extended such that each work item $w$ has a value point score, noted $valuePoints(w)$, denoting the relative value of $w$ in comparison to other work items. The 'Business Value' attribute used in project management systems such as JIRA or Azure DevOps are typical example of such value points. Value points are provided by release planners, product owners or project stakeholders.

The values points of a release $i$ in a plan $p$ is the sum of the value points for all work items planned for that release:

$$valuePoints(p,i) = \sum_{\{w \in WI \mid p(w)=i\}} valuePoints(w) \tag{2}$$

The value points of release plan $p$ is the weighted sum of the value points in each release:

$$valuePoints(p) = \sum_{i=1}^{H} weight(i) \times valuePoints(p,i) \tag{3}$$

where $weight(i)$ denotes the weight given to value points in release $i$. The release weights, to be specified by the release planners, denote the relative importance of value points delivered in earlier releases over those delivered in later releases. Formally, the weights define marginal rates of substitution between value points in different releases. For example, if the first three releases have weights of 10, 5 and 1, respectively then according to Equation 3, a single value point delivered in the first release has the same value as 2 value points delivered in the second release, and the same value as 10 value points delivered in the third.

The release planning method EVOLVE-II extends this model by deriving a work item's value point from the assessment of multiple criteria by multiple stakeholders [61]:

$$valuePoints(w) = \sum_{c \in C} weight(c) \times \sum_{x \in S} (weight(x) \times score(w,x,c)) \tag{4}$$

where

- $C$ is a set of criteria and *weight(c)* is the weight of criteria $c$;
- $S$ is a set of stakeholders and *weight(x)* the weight of stakeholder $x$;
- *score(w, x, c)* denotes the score assigned by stakeholder $x$ to work item $w$ for criteria $c$.

For example, Table 1 illustrates how value points for three features in our motivating example would be elicited using EVOLVE-II. The two criteria of interests are 'Frequency' denoting how often a service is used and 'Savings' denoting how much savings would be made by delivering this

Table 1. Eliciting value points in EVOLVE II

| Work Item | Frequency | | Savings | | ValuePoints |
|---|---|---|---|---|---|
| | Residents | Staff | Residents | Staff | |
| A: view council tax bills | 5 | 4 | 2 | 6 | 4 |
| B: apply for council tax reduction | 7 | 5 | 3 | 8 | 5 |
| C: pay council tax | 5 | 5 | 6 | 9 | 7 |

**Stakeholders Weights:** Residents = 0.6, Staff = 0.4
**Criteria Weights:** Frequency = 0.3, Savings = 0.7

feature. The two stakeholders groups are the local council residents and staff. Each stakeholders group is asked to score each work item against each criteria on a scale from 0 to 9. The work items' value points are then computed according to Equation 4.

Optimisation algorithms can then be used to shortlist a small set of release plans that maximize value points (Eq. 3) subject to satisfying the capacity constraints (Eq. 1). Release planners can then inspect the shortlisted plans and select their preferred one. A natural shortlisting strategy is to select the top $n$ release plans that satisfy the capacity constraints, for example with $n = 10$. A variant, used by EVOLVE-II, is to apply heuristics to increase diversity among the selected release plans while keeping the selected plans within a given bound of the best plan [61, 62].

Surveys have counted at least 22 variants of EVOLVE (16 up to 2010 [68] and an additional 6 up to 2016 [5]). Most variants amounts to pre-selecting the criteria $C$ to be used in Equation 4. Examples of such criteria include 'Urgency', 'Risk', 'Stakeholder satisfaction', 'Competitiveness', and 'Cost of delay'. All such criteria are evaluated as abstract scores and aggregated through weighted sums, although some variants use more complex formulae. One noteworthy extension consists in evaluating release plans from the perspective of each stakeholder group separately in order to analyse conflicts and fairness in the release planning process [23].

An important but often overlooked limitation of evaluating value points is the difficulty of eliciting scores and weights that accurately reflect the stakeholders' true preferences. In Equations 3 and 4, the weights and scores correspond to marginal rates of substitution. Eliciting such rates of substitution is far from trivial and one can question whether the elicited numbers (such as those in Table 1) accurately reflect the stakeholders' and release planners' true preferences. With inaccurate inputs, the evaluation and ranking of release plans may also be inaccurate.

## 3.4 Optimizing Net Present Value

An alternative to optimising value points is to optimise financial metrics such as net present value and return on investment [14, 19, 20, 69]. The Incremental Funding Method (IFM) is the first method to use such economic approach [19, 20]. It requires release planners to estimate the projected cash flow of each work item, $\mathit{cashFlow}(w) : [1..L] \to \mathbb{R}$ where $L$ is the investment horizon ($L \geq H$), i.e. the period over which the total value of release plans is measured, and $\mathit{cashFlow}(w, i)$ denotes the projected cost or revenue of $w$ during the $i^{th}$ period after the start of its development. For example, $\mathit{cashFlow}(w) = [-2000, 1000, 1000, 1000, 1000]$ means that developing $w$ will cost \$2,000 during one period and once released will generate \$1,000 per period for the next four periods.

In the IFM, value is more easily defined over development plans than release plans. A development plan is a partial function $p : \mathit{WI} \to [1..H]$ that maps work items to the period in which their development starts. A development plan's cash flow is a function $\mathit{cashFlow}(p) : [1..L] \to \mathbb{R}$ computed from the work items' projected cash flow such that $\mathit{cashFlow}(p, i)$ is the sum of all work items' costs and revenues taking into account when their development started. The Net Present Value (NPV) of a development plan $p$ is then defined as:

$$NPV(p) = \sum_{i=1}^{L} \frac{cashFlow(p, i)}{(1+r)^i} \tag{5}$$

where $r$ is the discount rate. NPV is a standard financial metric to compare cash flows taking into account the time value of money at a given discount rate. The discount rate allows one to take into account that \$100 dollars today are worth more than \$100 dollars in a year. Note that the discount rate plays a similar role as the release weights in the previous section; it allows one to compare values delivered at different times.

Instead of including capacity constraints as defined in Section 3.2, the IFM assumes the development team can only work on a bounded number of work items per period (e.g. at most 2 work items per period). This assumption is admittedly too strong and unrealistic in many situations. The IFM uses heuristics to search for development plans that optimise NPV. The output of the IFM is a single development plan that has optimal or near optimal NPV. Release planners can use this plan as baseline to explore alternatives and select some preferred release plan.

Extensions to the IFM include improved algorithms for identifying optimal development sequences [3], analysis of cash flows uncertainty [10, 58], and analysis of competitors' behaviours using game theory [48]. The IFM extensions dealing with cash flow uncertainty ignore uncertainty about development time and assume fixed-scope releases. They also assume a single work item can be worked on at a time. The new release planning method in Section 4 will address these limitations.

Arguments against such economic analysis are the inaccuracy of most cost and revenue projections and their tendency to overlook important but not easily quantifiable values, for example stakeholders' satisfaction and other human, social and environmental values. An advantage over optimizing value points, however, is that economic values can often be derived from observable factors. For example, in our local government project, the economic value of a feature can be evaluated by estimating how many phone interactions per months will be saved by introducing the feature and multiplying it by the average cost of a phone interaction. We argue such evaluation is more credible (or at least less arbitrary) than the weighted sum of scores between 1 and 9 in Table 1. Values that at first may seem hard to quantify, such as the residents satisfaction in our example, can also often be related to observable factors (e.g. residents' praises and complaints about the council's services) and measured in economic terms by eliciting stakeholders' *willingness-to-pay* for such values [29]. Such economic analysis are nevertheless fraught with uncertainty. The next section presents a novel release planning method that takes such uncertainty into account.

## 4 RELEASE PLANNING WITH BEARS

BEARS supports release planning under uncertainty in the context of fixed-date release cycles. The main input to BEARS is a product backlog where each work item $w$ has the following attributes:

- *effort(w)*: a real-valued probability distribution denoting the release planners' subjective uncertainty about the number of person-days required to deliver $w$;
- *value(w)*: a real-valued probability distribution denoting the release planners' subjective uncertainty about the value in monetary units brought about by $w$ in each period after it is delivered.

BEARS also requires release planners to specify the following additional parameters:

- the planning horizon $H \in \mathbb{N}^+$ and the team capacity, noted *capacity(i)*, in each period $i \in [1..H]$;

- the investment horizon $L \in \mathbb{N}^+$ and the discount rate $r \in \mathbb{R}$ used to compute net present values;
- and, optionally, the budget $B$ allocated to development team for the next $H$ iterations. This budget is involved in the computation of net present value. If no budget is specified, $B = 0$.

Given these inputs and parameters, BEARS evaluates candidate release plans against the following criteria:

- their *Expected Net Present Value (ENPV)*, and
- their *expected punctuality* (EP) defined as the expected percentage of work items delivered on time;

BEARS is supported by a tool that automatically shortlists a set of Pareto-optimal release plans that maximize expected NPV and expected punctuality. The tool is implemented in JAVA. It uses JMetal [55] for multi-objective optimisation and our implementation of a Monte-Carlo simulation of release plans to be described in Section 4.2. The tool takes as input a CSV file defining the work items' precedence relation and effort and value estimates. It generates a CSV file with the shortlisted Pareto-optimal release plans and a figure of the Pareto front.

*Notes.*

(1) BEARS does not assume that a feature will be developed within a single iteration; the development of a feature can start in one iteration and end in a later iteration. BEARS also does not assume that the total effort to develop a feature is smaller than the team capacity in each iteration. Good practices of iterative development recommend avoiding such large features, but the mathematical model in BEARS allows them. Unlike methods that ignore uncertainty, the BEARS model does not include a capacity constraint (Eq. 1) that is incompatible with such large features.

(2) The budget parameter $B$ is necessary to compute net present value. BEARS assumes a simple model where the development team is allocated a fixed budget for its time and material for the next $H$ releases; this budget is independent of the release plan. It corresponds to the fixed salary and expenses of the development team over the planning horizon. If needed, the method can be extended to support more complex cost models, for example where the budget varies in each iteration or depends on the features being developed. The required changes are described in Section 4.2.2. Release planners can choose to ignore the budget by leaving the model parameter $B$ unspecified. The tool will then compute net present values using $B = 0$.

(3) By choice, BEARS does not model uncertainty about team capacity. Rather, its model assumes the team capacity for each period to be known and fixed. This is a simplification because the team capacity could vary due to uncertain events (e.g. sickness, people leaving the team, etc.). In practice, these uncertain events are usually not considered during strategic release planning. They are rather managed during operational release planning (i.e. during each iteration), for example by adapting the release plan [1] or by recovering lost capacity with overtime [22]. We have kept the BEARS model in line with such practices. If needed, BEARS can be extended to deal with uncertainty about team capacity. This involves simple changes described in Section 4.2.1.

## 4.1 Eliciting Effort and Value Uncertainty

Reliable, principled techniques exist for eliciting subjective uncertainty from persons and groups of persons [56]. These techniques assume that people have some real, tacit uncertainty about the quantities of interest (in our case, the work items' effort and value) that can be uncovered through targeted questions and modelled as Bayesian probability distributions. The techniques are designed to mitigate common estimation biases such as over-confidence and anchoring.

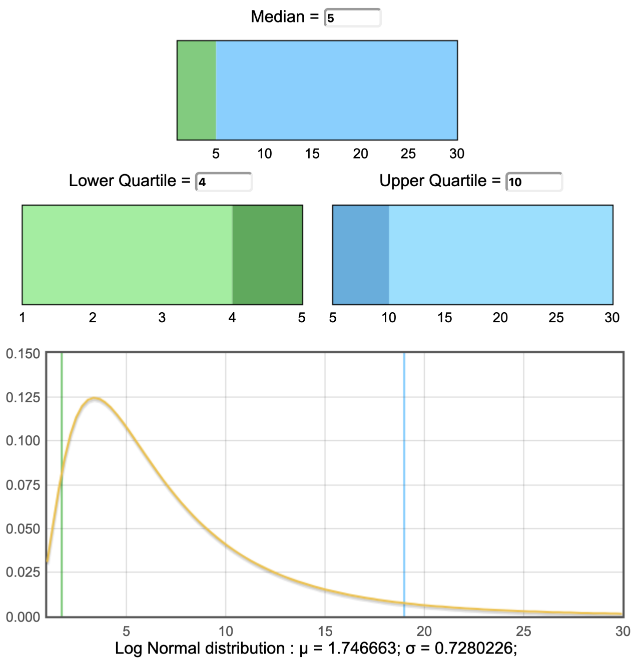


Fig. 2. Eliciting effort uncertainty for feature “A: view council tax bills” with the MATCH tool [52].

We use the SHELF elicitation framework [56] and its associated MATCH tool [52] to perform such elicitation. SHELF is a comprehensive framework that includes a wide range of elicitation techniques. In this paper, we use the ‘quartile method’ to elicit uncertainty about work items’ effort and value. The method is described below. We also assume that uncertainty about work items effort and value have lognormal distributions. Lognormal distributions are appropriate to model uncertainty about quantities that are always positive and where the distribution can be asymmetric and have a long tail of possible high values [30, 56]. An alternative to lognormal distributions is to use triangular distributions created by eliciting three point estimates: the worst, most likely, and best cases [41, 42, 46, 63]. This approach, however, has been criticized for its failure to elicit uncertainty accurately [56, 57]. Although BEARS currently uses lognormal distributions, its simulation method in Section 5 can be used with any probability distribution, including triangular distributions.

We describe the quartile method by illustrating its application for effort estimation. Eliciting uncertainty about work items’ value is similar. For example, Figure 2 illustrates the elicitation of effort uncertainty for work item A in the product backlog. The quartile method consists in eliciting the development team’s subjective assessment of: (i) the effort’s *upper and lower limits* denoting minimum and maximum plausible values; (ii) the effort’s *median* value denoting the value $M$ such that the development team judges it equally likely that the true effort is below $M$ or above $M$; (iii) the effort’s *lower quartile* denoting the value $Q1$ such that the development team judges it equally likely that the true effort is below $Q1$ or between $Q1$ and $M$; and (iv) the effort’s *upper quartile* denoting the value $Q2$ such that the development team judges it equally likely that the true effort is above $Q2$ or between $M$ and $Q2$. A facilitator helps team members adjust their estimates by testing their indifference to alternative bets. For example, to test the median, the facilitator asks the development team to consider a bet where they would win a large payout if they can correctly guess whether the true effort will be either below or above the stated median. If they have a preference for betting either ‘below’ or ’above’, then their stated median is not the true median of their subjective uncertainty and should be adjusted accordingly. Once all data points have been elicited, the MATCH tool infers the best-fit lognormal distribution as shown in Figure 2. Throughout the paper, all estimates for the local government project have been defined by the paper’s authors for illustration purpose.

## 4.2 Simulating and Evaluating Release Plans

To simulate and evaluate release plans, we distinguish *release plans* $p : WI \longrightarrow [1..K]$ that specify when work items are planned for release from *release scenarios* $s : WI \longrightarrow [1..K]$ that specify when work items are actually released. During release planning, the actual release scenario is unknown.

BEARS evaluates release plans’ ENPV and expected punctuality using Monte-Carlo simulation. The approach consists in simulating multiple release scenarios of a release plan, evaluating the net present value and punctuality of each scenario, and estimating ENPV and expected punctuality as the mean net present value and mean punctuality over all simulated scenarios. This approach, whose high-level pseudo code is shown in Figure 3, is described in more details below.

*4.2.1 Simulating Release Scenarios.* To evaluate release plans, BEARS creates a `ReleasePlan-Simulator` associated to the product backlog. The function `createReleasePlanSimulator` in Figure 3 generates $N$ simulations of work items’ effort and value drawn from the effort and value probability distributions. By default, BEARS uses $N = 10^4$. This creates $N$ possible future worlds, each having specific effort and value data for each work item. In Figure 3, the variables *effort* and *value* are matrices of size $N \times |WI|$ such that *effort[n,w]* and *value[n,w]* denote the simulated effort and value of work item $w$ in the $n^{th}$ simulation, respectively. Note that BEARS’s choice of probability distribution is hidden in the function `createReleasePlanSimulator`; extending the `ReleasePlanSimulator` to deal with different probability distributions than lognormal would only requires changing the implementation of this function.

In each possible future world, the function `generateReleaseScenario` creates the release scenario that would happen in this world based on the work item’s effort data. Note that release plans do not specify any priority between work items to be delivered in the same release. However, in order to generate a release scenario, we need to make some assumption about the order in which work items will be worked on so that we can identify what work items are most likely to be postponed to the next release if they cannot all be developed in time. Therefore, the function `generateReleaseScenario` first creates a work sequence *ws* that specifies in what order we assume work items will be completed, then generates the release scenario from the work sequence.

To generate a work sequence from a release plan, we assume the following priorities between work items: an item $w_i$ takes precedence over an item $w_j$ in the work sequence if: (i) $w_i$ is planned for an earlier release than $w_j$ (i.e. $p(w_i) < p(w_j)$), or (ii) $w_j$ has a precedence dependency on $w_i$ (i.e. $w_i \longleftarrow w_j$), or (iii) both items are planned for the same release and have no precedence dependency and $w_i$ has a higher ratio of mean value to mean effort than $wj$. These priorities are described in the `startEarlier` function in Figure 3.

From the work sequence, the loop in the function `generateReleaseScenario` assigns work items to release periods such that the $k^{th}$ item in the work sequence is delivered in release $i$ if, and only if, the cumulative effort to develop the work sequence up to item $k$ is more than the cumulative team capacity up to period $(i-1)$ and less or equal to the cumulative team capacity up to period $i$:

$$\sum_{j=1}^{i-1} capacity(j) < \sum_{j=1}^{k} effort(ws(j)) \leq \sum_{j=1}^{i} capacity(j) \quad (6)$$

```
Module ReleasePlanSimulator
    Internal Data:
        backlog: Backlog, including additional parameters $H$, capacity, $L$, $r$ and $B$.
        $N$: Integer initialized to $10^4$.
        effort: Matrix to hold work item effort simulation data.
        value: Matrix to hold work item value simulation data.

    Function createReleasePlanSimulator(Backlog b):
        Result: Generates $N$ simulations of work item's effort and value to be used for
                subsequent evaluation of release plans.
        $backlog \leftarrow b$
        for each work item w in b do
            effort[ _ , w] ← vector of $N$ random data drawn from $w$'s effort distribution
            value[ _ , w] ← vector of $N$ random data drawn from $w$'s value distribution
        end

    Function evaluateReleasePlan(ReleasePlan p):
        Result: a pair of value $< ENPV(p), expectedPunctuality(p) >$
        $sumNPV, sumPunctuality \leftarrow 0$
        for $n \leftarrow 1$ to $N$ do
            $s \leftarrow$ generateReleaseScenario (n, p)
            $sumNPV \leftarrow sumNPV + NPV(s)$          // $NPV(s)$ defined in Eq.7
            $sumPunctuality \leftarrow sumPunctuality + Punctuality(s, p)$
                                                  // $Punctuality(s, p)$ defined in Eq.10
        end
        return $< sumNPV \div N, sumPunctuality \div N >$

    Internal Function generateReleaseScenario(Integer n, ReleasePlan p):
        Result: the release scenario $s$ in the $n^{th}$ simulation.
        ws ← list of $p$'s work items ordered by startsEarlier(_, _, n, p)
        $s \leftarrow$ an empty release scenario
        $i \leftarrow 1$; $SumCapacity \leftarrow capacity$(i); $SumEffort \leftarrow 0$
        for $j \leftarrow 1$ to length(ws) do
            $SumEffort \leftarrow SumEffort + effort[n, ws(j)]$
            while $(SumEffort > SumCapacity)$ do
                $i \leftarrow i + 1$
                $SumCapacity \leftarrow SumCapacity + capacity(i)$
            end
            $s(i) \leftarrow s(i) \cup \{ws(j)\}$
        end
        return s
    Internal Function startsEarlier(WorkItem $w_i$, $w_j$, Integer n, ReleasePlan p):
        if $(p(w_i) < p(w_j)$ OR $w_i \leftarrow w_j$ in the backlog precedence relation) return TRUE
        else return $(value[n, w_i] \div effort[n, w_i] > value[n, w_j] \div effort[n, w_j])$
```

Fig. 3. High-level pseudo code for simulating and evaluating release plans under uncertainty.

where $ws(j)$ is the $j^{th}$ element in the work sequence $ws$. This condition ensures that the total effort in each release period does not exceed the team capacity for that period and that work items are released in order of the work sequence.

If BEARS was to be extended to analyse uncertainty about team capacity, the only changes required in Figure 3 would be to generate $N$ simulations of the team capacity for each iteration in `createReleasePlanSimulator` and store them as internal data, then change the two occurences of capacity(i) to capacity[n, i] in `generateReleaseScenario`.

Note that simulating release plans in the context of fixed-date release cycles is substantially different (and more complex) than under the assumption of fixed-scope releases. Methods that assume fixed-scope releases (e.g. [10, 46, 49, 58, 63]) start like BEARS by generating $N$ simulations of work items' effort and value from their probability distributions. The next steps are different. Assuming fixed-scope releases means that release scenarios do not deviate from the release plan (for all work items $w$, $s(w) = p(w)$) and only the release dates are uncertain. Simulated release dates are derived from the work items' simulated effort data: in each of the $N$ possible future worlds, the simulated date of the $i^{th}$ release is simply the sum of the simulated efforts of all work items to be delivered up to the $i^{th}$ release.

*4.2.2 Evaluating Net Present Value.* In a release scenario $s$, the value of work item $w$ in period $i$, noted *value(w, s, i)*, equals $value(w)$ if $w$ was released before period $i$ (i.e. $s(w) < i$), or 0 otherwise. The total value delivered during period $i$ of $s$ is the sum of the value delivered by all work items: $value(s, i) = \sum_{w \in WI} value(w, s, i)$. The Net Present Value (NPV) of release scenario $s$ is:

$$NPV(s) = \sum_{i=1}^{L} \frac{value(s,i)}{(1+r)^i} - B \tag{7}$$

where $B$ is the allocated budget for the planning horizon.

The net present value of a release plan $p$ is a random variable whose distribution is approximated by the simulated scenarios' net present values. The expected NPV of a release plan is the mean NPV over that distribution:

$$ENPV(p) = E[NPV(s)] \tag{8}$$

BEARS computes *ENPV(p)* as the mean of *NPV(s)* over all release scenarios simulating $p$. In addition to computing the expected NPV, BEARS could also compute other statistics about NPV such as the loss probability (the probability that the net present value is negative) and the Value-at-Risk defined as the $5^{th}$ quantile of the net present value probability distribution.

If BEARS was to be extend with a more complex cost model, for example where the cost of a release plan depends on the features developed rather than being a fixed budget, the only change required would be to modify the second term in Eq. 7.

*4.2.3 Evaluating Expected Punctuality.* The punctuality of a release scenario $s$ with respect to a release plan $p$ is defined as the percentage of planned work items delivered on or before their planned release period:

$$Punctuality(s, p) = \frac{\#\{w \in WI \mid s(w) \leq p(w)\}}{\#dom(p)} \tag{9}$$

where *dom(p)* is the domain of $p$, i.e. the set of work items in the release plan. This punctuality metric is similar to that used in other domains, notably in transport. For example, the punctuality of a railway line is the percentage of trains that arrived at their final destination on time.

The expected punctuality of a release plan $p$ is the mean punctuality of the $N$ release scenarios that simulate $p$:

$$EP(p) = E[Punctuality(s,p)] \tag{10}$$

A release plan with a 90% expected punctuality means that 90% of work items are expected be delivered no later than their planned release. This expected punctuality metric is motivated by the need for a simple metric to help release planners communicate the uncertainty associated with different release plans to clients and other stakeholders. The metric allows release planners to compare release plans with different expected punctuality and to analyse tradeoffs between expected punctuality and expected NPV, as it will be explained in Section 4.4.

*4.2.4 Computational Complexity.* The function `generateReleaseScenario` involves sorting the work items in the release plans and has thus a worst-case theoretical complexity of $|p| \ln |p|$ where $|p|$ is the number of work items in the release plan $p$. The function `evaluateReleasePlan` involves generating and evaluating $N$ release scenarios and therefore has a computational complexity of $N \times |p| \ln |p|$. The computational cost of evaluating release plans under uncertainty is therefore $N \times |p| \ln |p|$ times larger than when uncertainty is ignored. Experiments in Section 5 will report the differences in run times on concrete release planning problems.

## 4.3 Shortlisting Optimal Release Plans

BEARS shortlists a set of Pareto-optimal solutions that maximize expected net present value (ENPV) and expected punctuality (EP). In BEARS, a release plan $p$ is Pareto-optimal if there is no other release plan $p'$ that outperforms $p$ for one criteria and is at least as good for the other criteria, i.e. such that either $ENPV(p') > ENPV(p)$ and $EP(p') \geq EP(p)$, or $EP(p') > EP(p)$ and $ENPV(p') \geq ENPV(p)$.

In general, the number of possible release plans will be too large to compute the exact set of Pareto-optimal solutions. BEARS thus relies on multi-objective evolutionary algorithms (MOEAs) to explore the space of candidate release plans and generate a good approximation of the set of Pareto-optimal plans. Our implementation provides a choice among three alternative MOEAs (users can choose one of the three): NSGA-II [17], MOCell [54], and SPEA2 [74]. We use these MOEAs because they have readily available implementations in JMetal, the optimisation framework used in BEARS [55], and they are among the most widely used MOEAs for release planning [73].

To apply such algorithms, BEARS encodes a release plan $p$ as an array of integers where each element represents the release number $p(w)$ assigned to a work item, or zero if the work item is not scheduled in the plan. The size of the array is equal to the number of the work items in the product backlog. The fitness of a release plan is evaluated using the `evaluateReleasePlan` function in Figure 3. The MOEA starts by randomly generating an initial population of 100 release plans, then iteratively evolve the population towards release plans with higher ENPV and EP using genetic selection, crossover and mutation. Our implementation uses integer simulated binary crossover with probability $P_c = 0.9$ and polynomial mutation with probability $P_m = \frac{1}{|WI|}$ where $|WI|$ is the number of work items. These are standard parameter settings used in previous research [73]. We have not tuned the parameters to optimise the MOEAs. The MOEAs could potentially be optimised through parameter tuning, by using hyperheuristics, or by devising better problem-specific algorithms (all are topics for future research). We have set the MOEAs to terminate after having explored $25,000$ release plans. On termination, they return the Pareto-optimal release plans in their final population.

During mutation and crossover, the MOEAs may generate candidate release plans that violate the precedence constraints between work items. Following an approach used in previous work [64], our implementation automatically detects and repairs such violations so that the populations only contain valid release plans. If in the future BEARS is extended to support more dependency relations

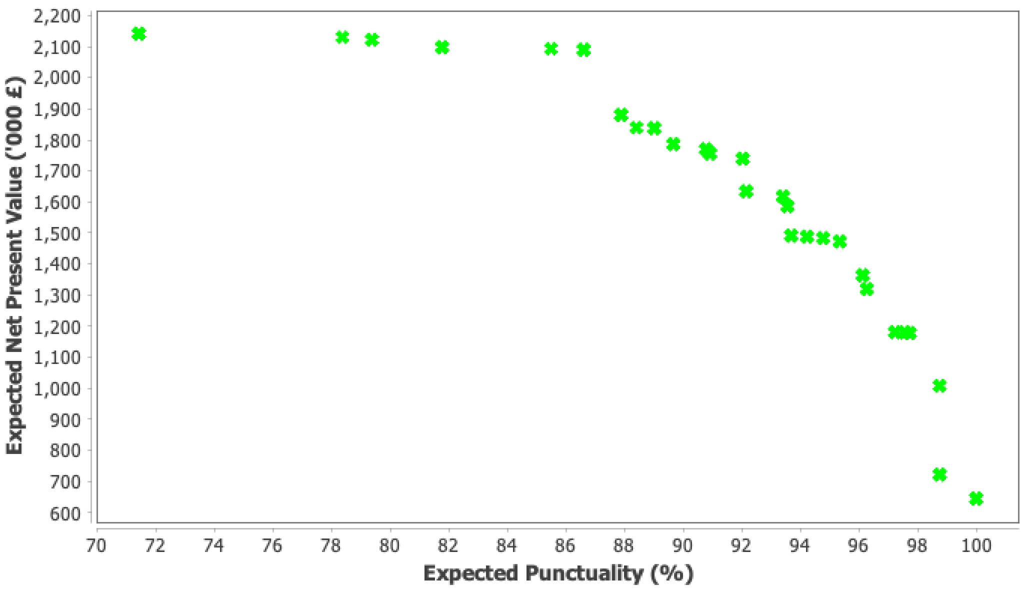


Fig. 4. Expected NPV and punctuality of shortlisted release plans for the local government project.

between work items, such as such as coupling, exclusion and weak precedence (see Section 3.1), the only required change will be to extend the detection and repair of constraints violations during mutation and crossover.

Note that an important feature of MOEAs is to consider diversity in their exploration and evaluation of candidate solutions [17, 54, 74]. Diversity of the shortlisted release plans in BEARS is therefore achieved through the MOEAs and the multiobjective nature of BEARS optimisation model. The next section will show an example of a diverse shortlist and how such diversity support release planning decisions.

The computational cost of this shortlisting step is the number of release plans being evaluated by the MOEA $(25,000)$ multiplied by the complexity of evaluating each release plans $(N \times |p| ln |p|)$. In our running example, BEARS takes on average 48 seconds to shortlist release plans with a planning horizon $H = 3$. The experiments in the Section 5 will measure and report run times for release planning problems of increasing sizes.

## 4.4 Visualising Shortlisted Release Plans

When the search terminates, BEARS displays the shortlisted release plans' Pareto-front. Visualising such Pareto-front helps release planners analyse trade-off between expected value and punctuality.

For example, Figure 4 shows the returned Pareto-front for our local government problem. The release plan with the highest ENPV, on the top left of the figure, has an expected punctuality around 70%. Release planners may hesitate to choose a release plan where 30% of work items are likely to be late. Late deliveries, even if forewarned, will disappoint stakeholders and could put the project at risk of being cancelled. Developers' morale and productivity could also suffer. To address these concerns, release planners could select a release plan with slightly lower ENPV but higher expected punctuality up to 86%. Release plans with still higher expected punctuality are possible but would required further sacrifice in ENPV. If the client demands a punctuality of 95% or even 100% (e.g.

with penalties if targets are not met), release planners can select a safer release plan, reducing ENPV to around £1,500K for 95% expected punctuality and to around £600K for 100%.

In summary, visualising the ENPV and expected punctuality of shortlisted release plans enables release planners to have informed discussions with the project clients and development team about the possible tradeoffs between ENPV and expected punctuality. These discussions are important: in every project, release planners need to define a release plan that strikes the right balance between being too ambitious (high expected net present value but low expected punctuality) and too conservative (high expected punctuality but low expected net present value). Analysing uncertainty using BEARS enables informed discussions about such tradeoffs.

## 5 EXPERIMENT: CAN UNCERTAINTY BE IGNORED?

In Sections 3 and 4, we have presented reasoned arguments why analysing uncertainty during release planning is *in theory* better than ignoring it. In a nutshell, analysing uncertainty allows for more realistic evaluation of release plans than if release plans are evaluated using inaccurate effort and value estimates that ignore uncertainty. Analysing uncertainty also enables informed discussions about tradoffs between expected value and expected punctuality that are not possible when uncertainty is ignored.

In this section, we study whether analysing uncertainty makes a difference in practice: If BEARS and methods that ignore uncertainty were applied in the same context, would they shortlist the same release plans? Ignoring uncertainty may not be so harmful if the shortlisted release plans are the same whether one analyses uncertainty or ignores it. The following experiment studies whether this is the case.

### 5.1 Experiment Design

Our research question is:

> Does BEARS shortlist better release plans than methods that ignore uncertainty?

To study this question, we apply BEARS to 32 release planning problems drawn from 8 product backlogs and we compare the release plans shortlisted by BEARS to release plans shortlisted by methods that ignore uncertainty. For this experiment, we consider that a release plan is better than another if it has higher expected NPV and expected punctuality (i.e. strict dominance in multiobjective optimisation [76]). In other words, we assume that BEARS optimisation criteria are consistent with the release planners' true preferences: if $ENPV(p_1) > ENPV(p_2)$ and $EP(p_1) > EP(p_2)$ then release planners prefer $p_1$ over $p_2$. This is a reasonable assumption that we will justify further in Section 5.1.6.

*5.1.1 Release Planning Problems.* Table 2 lists the 8 product backlogs in our experiment. For each product backlog, we consider 4 release planning problems by varying the planning horizon from 2 to 5 periods. The local government project is the illustrative example used throughout the paper. The Release Planner, Word Processor, and RALIC product backlogs originate from previous studies and have been used extensively in previous research [7, 59, 71–73]. The first two contain candidate features for future releases of a commercial release planning tool (Release Planner) and a word processor, respectively. The features were elicited during exploratory studies for a variant of the EVOLVE method [35]. The product backlog for the RALIC project includes requirements for a building access control system at University College London. A team of researchers (other than this paper's authors) elicited the RALIC requirements from 87 stakeholders using an online tool that leverage stakeholders' relationships to drive the requirements elicitation and prioritization process [44]. The dataset has no constraint because no constraints were elicited during this elicitation

Table 2. Release Planning Problems in the Experiment

| Release Planning Problem | Backlog Size | Number of Constraints | Original effort estimates | Original value estimates |
|---|---|---|---|---|
| Local Government Project | 20 | 23 | uncertain person-days | uncertain economic value |
| Release Planner | 25 | 11 | person-hours | value points |
| Word Processor | 50 | 65 | person-hours | value points |
| RALIC | 143 | 0 | person-hours | value points |
| Synthetic-30 | 30 | 11 | uncertain person-days | uncertain economic value |
| Synthetic-50 | 50 | 15 | uncertain person-days | uncertain economic value |
| Synthetic-100 | 100 | 30 | uncertain person-days | uncertain economic value |
| Synthetic-200 | 200 | 51 | uncertain person-days | uncertain economic value |

Table 3. Release Planning Methods in the Experiment

| Model | Effort estimates | Value estimates | Simulation Method | Optimisation problem |
|---|---|---|---|---|
| BEARS | uncertain man-days | uncertain economic value | stochastic, flexible-scope | Maximise expected NPV; Maximise expected punctuality |
| NPV-*deterministic* | man-days | economic value | deterministic | Maximise NPV subject to capacity constraint |
| VP-*deterministic* | story points | value points | deterministic | Maximise value points subject to capacity constraint |

process. The last 4 product backlogs are synthetic backlogs of size 30, 50, 100, and 200, respectively. The work items precedence relations, effort and value estimates have been generated at random.

The original effort and value estimates for the Release Planner, Word Processor and RALIC product backlogs do not include uncertainty. In order to use these problems in our experiment, we have artificially added uncertainty to these estimates by simulating the uncertainty elicitation process described in Section 4.1. For an original effort estimate $x$, we have set the lower and upper bounds to $x$ and $1.8x$ and the lower quartile, median and upper quartile to $1.2x$, $1.5x$ and $1.7x$. This simulates a situation where the original effort estimate is the most optimistic value and the true development time could take up to 1.8 times this optimistic value. This situation is consistent with studies of software estimations that show that people tend to underestimate development time [32]. For an original value point estimate $y$, we have set the lower and upper bounds to 0 and $1000y$ and the lower quartile, median and upper quartile to $200y$, $500y$, and $750y$. This simulates a situation where a single value point is worth $\$1,000$, the original value estimate is the most optimistic value and the true value could be as low as zero. This is consistent with observations of software project where initial prediction of business value tend to be overestimated [24, 53]. We make no claim that the values we have chosen represent the true uncertainty that would have been elicited from real developers and stakeholders in these projects. Since our objective is to compare the outputs of different release planning methods when applied on the same inputs, it is sufficient that these inputs are realistic (in the sense that they are consistent with empirical studies [24, 32, 53]) and used consistently in our experiments.

*5.1.2 Release Planning Methods.* Table 3 summarizes the release planning methods in our experiment. BEARS was defined in Section 4. NPV-*deterministic* and VP-*deterministic* are deterministic variants of BEARS intended to be representative of release planning methods that optimise net present value and value points, respectively. We have designed these methods specifically for this experiment in order to be able to study the effect of analysing uncertainty in isolation from other differences between release planning methods.

NPV-*deterministic* takes the same inputs as BEARS except that work items' effort and value estimates are single numbers instead of probability distributions. The optimisation model is to maximize net present value (Eq. 5) subject to capacity constraints (Eq. 1). NPV-*deterministic* is representative of release planning methods such as the IFM [19, 20] that optimise net present value.

NPV-*deterministic* can be seen as an improved variant of the IFM. It differs from the IFM by including effort capacity constraints instead of relying on the unrealistic assumption that the development team can only work on a single work item in each iteration (see Section 3.4). Furthermore, it shortlists $n$ release plans using the same evolutionary optimisation algorithm as BEARS, whereas IFM shortlists a single release plan using heuristic rules. Comparing BEARS to NPV-*deterministic* instead of the IFM therefore allows us to study the effect of analysing uncertainty without being affected by other differences between BEARS and the IFM.

VP-*deterministic* takes as input work items' effort and value estimates given as story points and value points. The optimisation model is to maximize value points (Eq. 3) subject to capacity constraints (Eq. 1). VP-*deterministic* is representative of release planning methods such as EVOLVE-II [61] that optimise value points. Its optimisation model subsumes the optimisation model of EVOLVE-II: work items' value points can be estimated directly using the 'Business Value' attribute in project management systems or, as in EVOLVE-II, computed from the estimation of multiple criteria from multiple stakeholders (Eq. 4). VP-*deterministic* shorlists $n$ release plans using the same MOEAs with repair as BEARS. This differs from EVOLVE-II that uses integer programming with specific heuristics to increase diversity among the shortlisted release plans [61, 62]. Comparing BEARS to VP-*deterministic* allows us to study differences between the optimisation models of BEARS and methods that optimise value points without being affected by other differences such as variations in optimisation algorithms or elicitation techniques.

*5.1.3 Defining Consistent Model Inputs.* For each release planning problem, we would like to compare the methods when they are applied in the same conditions. In particular, we would like stakeholders to be consistent when they estimate the work items' effort and value. For our experiment, we thus define the NPV-*deterministic* and VP-*deterministic* estimates to consistent with the BEARS estimates. Given the BEARS probability distributions for effort and value, we set the NPV-*deterministic* estimates to be the mean of the corresponding probability distribution in BEARS. We then set the VP-*deterministic* estimates to be the mean of the BEARS probability distribution normalised on a scale from 0 to 9 (the default scale in EVOLVE-II). For the VP-*deterministic* model, we define the team capacity and release weights to be consistent with the team capacity and discount rate in the BEARS model: the team capacity in each iteration (used in Eq. 1) is set using the same conversion from person-days to story points used for converting BEARS effort estimates to story points; the weight of each release period $i$ (used in Eq. 3) is set to be $(1 + r)^{-i}$ where $r$ is the NPV discount rate in the BEARS model. This ensures that both models apply the same relative weights to sum up values in different release periods.

*5.1.4 Comparing Shortlists.* Once the models' inputs have been defined, we apply each method in turn to generate their shortlists. We have used NSGA-II as MOEA for all three methods. In BEARS, the shortlist size is determined by the number of Pareto-optimal release plans returned by the MOEA. In NPV-*deterministic* and VP-*deterministic*, the shortlist size $n$ can be chosen by the release planners. The default shortlist size in ReleasePlanner 2.0, the commercial software for EVOLVE-II method, is 10. We therefore decided that the the shortlist size for NPV-*deterministic* and VP-*deterministic* should never be smaller than 10 and, for a fair comparison, should always be at least as large as the BEARS shortlist. We have therefore set the shortlist size for NPV-*deterministic* and VP-*deterministic* to be the maximum of 10 and the BEARS shortlist size. (It later turns out that in our experiment all BEARS shortlists are larger than 10.) We then compute the expected NPV and expected punctuality of all shortlisted release plans using the function `evaluateReleasePlan` in Figure 3. Note that this function can be applied to release plans shortlisted by NPV-*deterministic* and VP-*deterministic* even though these release plans were shortlisted using different criteria.

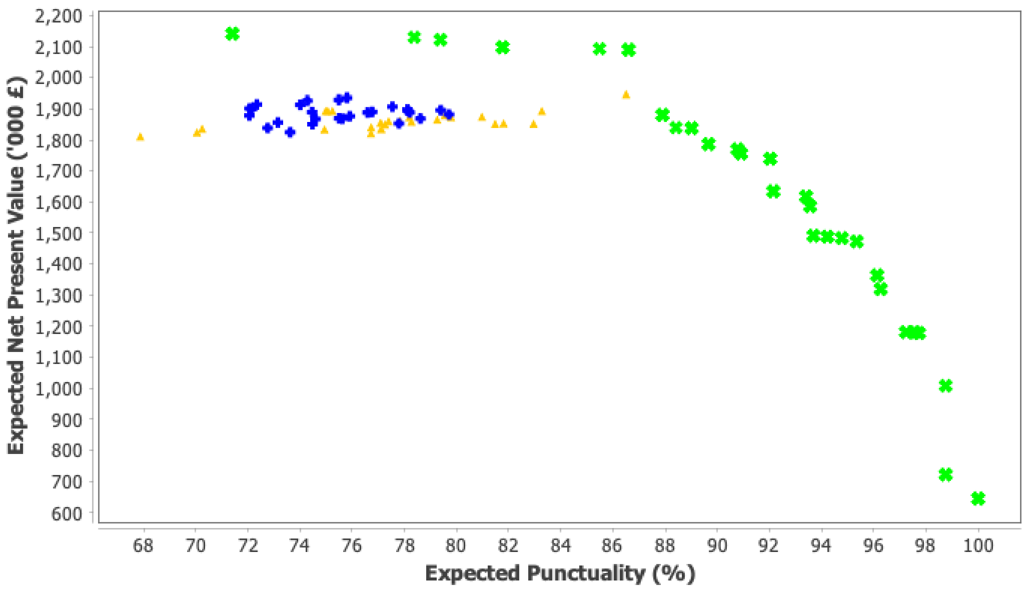


Fig. 5. Release plans shortlisted by BEARS (green crosses), NPV-*deterministic* (yellow triangles) and VP-*deterministic* (blue diamonds) for the local government project.

*5.1.5 Illustrative Example.* Figure 5 shows the result of comparing shortlists for our local government project. Figure 5 extends Figure 4 (shown in Section 4.4) by showing the ENPV and expected punctuality of release plans shortlisted by NPV-*deterministic* and VP-*deterministic* alongside those shortlisted by BEARS. The figure shows that:

(1) The shortlists are disjoint; none of the release plans shortlisted by BEARS are shortlisted by NPV-*deterministic* and VP-*deterministic*.
(2) The BEARS shortlist *strictly dominates* the two other shortlists; in other words, for every release plan in the NPV-*deterministic* and VP-*deterministic* shortlists there is at least one release plan in the BEARS shortlist that has higher expected NPV and higher expected punctuality.
(3) The maximum expected NPV is nearly 10% higher in the BEARS shortlist.
(4) The expected punctuality in the BEARS shortlist ranges from 72% to 100%. In contrast, the highest expected punctuality in the NPV-*deterministic* and VP-*deterministic* shortlists is 87% and 80%, respectively. Most importantly, in practice users of methods that ignore uncertainty will not know the release plans' expected punctuality.

Investigating the causes of these results provides interesting insights into subtle harmful effects of ignoring uncertainty during release planning.

That BEARS shortlists release plans with higher expected punctuality is not surprising given that NPV-*deterministic* and VP-*deterministic* ignore expected punctuality. Without any information about the uncertainty of effort estimates, such methods cannot evaluate expected punctuality. Instead of evaluating expected punctuality, these methods require that release plans satisfy the capacity constraint (Eq. 1). They then assume that all work items will be delivered on time, which amounts to assuming 100% punctuality. They are therefore unable to compare release plans' expected punctuality. Furthermore, optimising for value (in terms of NPV or value points) is detrimental to expected punctuality because release plans with the most value tend to use as much of the team's capacity as possibly and are therefore at greater risk of exceeding the team's capacity during development. Conversely, release plans with high expected punctuality, such as those with more than 90% expected punctuality in Fig. 5, are not shortlisted by NPV-*deterministic* and VP-*deterministic* because they have lower NPV and value points than release plans that pack more work items per iterations.

That BEARS also shortlists release plans with nearly 10% higher expected NPV is more surprising because all three methods optimize for value using related metrics: ENPV, NPV and value points. Moreover, our experiment uses consistent model inputs so at to minimize differences between the three methods. It is therefore not unreasonable to expect that all three methods would be more or less consistent in their ranking of release plans by value, i.e. that the top value release plans according to NPV-*deterministic* and VP-*deterministic* would also be top value, or close to top value, according to BEARS. However, this is not the case for the release plans in Fig. 5: the top value release plans according to NPV-*deterministic* and VP-*deterministic* have higher NPV and value points than the 6 BEARS release plans above £2,000K but much lower ENPV (lower by 10%). This is due to NPV-*deterministic* and VP-*deterministic* evaluating a release plan's value under the invalid assumption that all work items will be delivered in their planned release. When some features are delivered later than planned, the actual NPV or value point of the release plan decreases because the feature starts generating value later than planned. Unlike BEARS, NPV-*deterministic* and VP-*deterministic* have no way of estimating the likelihoods of late feature delivery. Not taking these likelihoods into account leads to incorrect and inconsistent evaluation of release plan values.

*5.1.6 Discussion about experiment validity.* Readers may wonder whether comparing release planning methods that have different optimisation models is valid: BEARS has two optimisation objectives, whereas NPV-*deterministic* and VP-*deterministic* only one. Comparing different optimisation models is the whole purpose of the experiment. In that regard, our experiment differs substantially from previous experiments in software release planning. In previous experiments, the objective is to compare the performance of different optimisation algorithms on software release planning problems (see [73] for the most comprehensive study to date). In these experiments, the optimisation model is a controlled variable (it must be fixed), and the optimisation algorithm is the independent variable. Our experiment is different. Our objective is to compare the methods' optimisation models, not their optimisation algorithms. The optimisation model is the independent variable, and the optimisation algorithm is a controlled variable.

Readers may then question the fairness of using the BEARS optimisation criteria (ENPV and EP) for defining what it means for a release plan to be better than another. Our experiment uses these criteria as proxy measures for the release planner's true preferences. As mentioned at the start of Section 5.1, this choice is based on the reasonable assumption that if $ENPV(p_1) > ENPV(p_2)$ and $EP(p_1) > EP(p_2)$ then release planners will prefer $p_1$ over $p_2$. We also claim that the BEARS criteria are better proxy measures for the release planners' true preferences than the criteria used by NPV-*deterministic* and VP-*deterministic* (NPV and value points, respectively). To support this claim, consider a situation where the criteria disagree, for example where $ENPV(p_1) > ENPV(p_2)$ and $EP(p_1) > EP(p_2)$ but $NPV(p_1) < NPV(p_2)$. This situation arises in Fig. 5: the release plans shortlisted by BEARS have higher ENPV and expected punctuality than those shortlisted by NPV-*deterministic* but they have lower NPV. In such situations, we argue that release planners prefer $p_1$ over $p_2$ despite $p_2$ having higher NPV because: (i) they will not discard information about expected punctuality revealed by BEARS, and (ii) they will regard BEARS evaluation of expected net present value (that takes uncertainty into account) as more reliable than the simpler evaluations of net present value (that ignore uncertainty). We therefore argue that BEARS optimisation criteria provide a better model of the release planners' true preferences than models that optimise net present value (or value points) without considering uncertainty. The purpose of the experiment is then to study how much ENPV and expected punctuality might be lost if release planners used NPV-*deterministic* or VP-*deterministic* instead of BEARS.

*5.1.7 Evaluation Metrics.* We use two metrics to quantify the differences between release plans shortlisted BEARS and other methods when they are applied to a set of release planning problems.

The first metric observes how often the BEARS shortlist *strictly dominates* the shortlists of other methods. In our context, a release plan $p1$ strictly dominates a release plan $p2$ if $ENPV(p1) > ENPV(p2)$ and $EP(p1) > EP(p2)$. A shortlist $L1$ strictly dominates a shortlist $L2$ if every release plan in $L2$ is strictly dominated by at least one release plan in $L1$. Strict dominance is the strongest form of dominance relation between sets of solutions in multi-objective optimisation problems [76]. If a single release plan in $L2$ is equal to, or is not dominated by some release plan in $L1$, then $L1$ does not strictly dominate $L2$.

Analysing strict dominance can tell us whether the BEARS shortlists are better than those shortlisted by other methods but it does not tell us *how much* better. Our second metric quantifies the improvement of the BEARS shortlist over the shortlist of another method by comparing their hypervolumes. In multi-objective optimisation problems, the hypervolume of a solution set $A$, noted *HV(A)*, is the volume of the objective space dominated by $A$ [75]. For example, in Figure 5 the hypervolume of a shortlist is the area dominated by the release plans in the shortlist, bounded below by the x-axis (*ENPV*=0) and to the left by the y-axis (*EP*=0). In this example, the HV of the BEARS and VP-*deterministic* shortlists are 2047 and 1541, respectively. The HV of the BEARS shortlist is larger because it covers a larger area. The Hypervolume improvement ratio (*HVIR*) of a solution set $A$ over a solution set $B$ is $HV(A)/HV(B)$. In Figure 5, the HVIR of the BEARS shortlist over the VP-*deterministic* shortlist is 2047/1541 = 1.33. Hypervolume is a widely used metric to compare solution sets of multi-objective optimisation problems. We have chosen this metric because it has as simple visual interpretation and is compatible with strict dominance, i.e. if $A$ strictly dominates $B$ then $HV(A)/HV(B) > 1$ [76].

*5.1.8 Experimental Set-Up.* The MOEAs used by the release planning methods in Table 3 are stochastic algorithms that may shortlist different release plans each time they are executed on a given problem. To account for such randomness, we have executed 30 independent runs of the three release planning methods on all 32 release planning problems. Each release planning method is thus executed 960 times ($30 \times 32$). In total, our experiment includes $2,880$ independent runs ($3 \times 960$), resulting in as many shortlists. All runs were executed on a single PC with Intel Core i5 CPU at 3.20GHz x 4 and 8GB of RAM.

*5.1.9 Statistics.* For statistical significance testing [25], our null hypothesis is that no difference exist between the hypervolumes of shortlists generated by BEARS and methods that ignore uncertainty. We compute p-levels using Mann-Whitney U-test, a nonparametric test recommended for our context [8]. By convention, a result is said to be statistically significant if its p-value is less than 5%. Statistical significance is often misinterpreted (even by experts [25]) and should not be confused with practical significance [16, 36, 50]. Evaluating practical significance is more meaningful. Our experiment evaluates practical significance by measuring the strict dominance and hypervolume improvement ratio of BEARS over methods that ignore uncertainty. The hypervolume improvement ratio can be viewed as a problem-specific measure of effect size. It provides a simple and meaningful way to compare shortlists generated by different methods. We report the mean and range (min and max values) of such ratios for all release planning problems.

Table 4. Strict Dominance and Hypervolume Improvement Ratio of BEARS over NPV-*deterministic* and VP-*deterministic*.

| | | BEARS vs. NPV-*deterministic* | | | BEARS vs. VP-*deterministic* | | |
|---|---|---|---|---|---|---|---|
| Product Backlog | H | Strict Dominance | HVIR Mean | HVIR Range | Strict Dominance | HVIR Mean | HVIR Range |
| Local Government Project | 2 | 21/30 | 1.27 | 1.22 −1.34 | 30/30 | 1.22 | 1.19 −1.26 |
| | 3 | 30/30 | 1.30 | 1.22 −1.55 | 30/30 | 1.24 | 1.20 −1.33 |
| | 4 | 27/30 | 1.32 | 1.19 −1.51 | 21/30 | 1.28 | 1.16 −1.76 |
| | 5 | 29/30 | 1.47 | 1.30 −1.68 | 27/30 | 1.34 | 1.18 −1.48 |
| Release Planner | 2 | 25/30 | 1.06 | 1.02 −1.11 | 23/30 | 1.05 | 0.97 −1.10 |
| | 3 | 30/30 | 1.08 | 1.05 −1.11 | 30/30 | 1.07 | 1.05 −1.10 |
| | 4 | 30/30 | 1.06 | 1.04 −1.09 | 30/30 | 1.06 | 1.04 −1.09 |
| | 5 | 30/30 | 1.09 | 1.06 −1.15 | 30/30 | 1.07 | 1.06 −1.10 |
| Word Processor | 2 | 24/30 | 1.01 | 0.96 −1.05 | 23/30 | 1.00 | 0.95 −1.05 |
| | 3 | 29/30 | 1.04 | 1.02 −1.06 | 29/30 | 1.04 | 1.02 −1.06 |
| | 4 | 30/30 | 1.05 | 1.04 −1.06 | 30/30 | 1.06 | 1.05 −1.07 |
| | 5 | 27/30 | 1.03 | 1.00 −1.04 | 30/30 | 1.04 | 1.01 −1.04 |
| RALIC | 2 | 30/30 | 1.12 | 1.08 −1.16 | 30/30 | 1.10 | 1.04 −1.20 |
| | 3 | 30/30 | 1.14 | 1.08 −1.22 | 29/30 | 1.08 | 1.02 −1.15 |
| | 4 | 30/30 | 1.16 | 1.11 −1.25 | 24/30 | 1.09 | 1.02 −1.14 |
| | 5 | 30/30 | 1.16 | 1.07 −1.26 | 22/30 | 1.08 | 1.01 −1.15 |
| Synthetic-30 | 2 | 30/30 | 1.11 | 1.05 −1.25 | 30/30 | 1.15 | 1.09 −1.67 |
| | 3 | 29/30 | 1.11 | 1.06 −1.20 | 30/30 | 1.14 | 1.09 −1.20 |
| | 4 | 30/30 | 1.15 | 1.10 −1.30 | 30/30 | 1.16 | 1.09 −1.23 |
| | 5 | 30/30 | 1.16 | 1.11 −1.22 | 30/30 | 1.16 | 1.12 −1.27 |
| Synthetic-50 | 2 | 29/30 | 1.14 | 1.03 −1.26 | 30/30 | 1.15 | 1.07 −1.30 |
| | 3 | 30/30 | 1.25 | 1.12 −1.42 | 30/30 | 1.23 | 1.10 −1.36 |
| | 4 | 30/30 | 1.30 | 1.22 −1.45 | 30/30 | 1.28 | 1.16 −1.42 |
| | 5 | 30/30 | 1.30 | 1.21 −1.40 | 30/30 | 1.30 | 1.20 −1.44 |
| Synthetic-100 | 2 | 30/30 | 1.08 | 1.05 −1.15 | 30/30 | 1.09 | 1.04 −1.14 |
| | 3 | 30/30 | 1.20 | 1.10 −1.33 | 30/30 | 1.21 | 1.14 −1.31 |
| | 4 | 30/30 | 1.32 | 1.20 −1.43 | 30/30 | 1.32 | 1.21 −1.44 |
| | 5 | 30/30 | 1.38 | 1.30 −1.51 | 30/30 | 1.39 | 1.28 −1.52 |
| Synthetic-200 | 2 | 30/30 | 1.11 | 1.05 −1.20 | 30/30 | 1.12 | 1.06 −1.22 |
| | 3 | 30/30 | 1.19 | 1.11 −1.26 | 30/30 | 1.17 | 1.10 −1.24 |
| | 4 | 30/30 | 1.26 | 1.18 −1.34 | 30/30 | 1.28 | 1.21 −1.38 |
| | 5 | 30/30 | 1.40 | 1.23 −1.50 | 30/30 | 1.38 | 1.24 −1.50 |
| **Overall** | | 97% (930/960) | 1.18 | 1.03 −1.42 | 96% (918/960) | 1.17 | 1.03 −1.39 |

## 5.2 Results

Table 4 shows the results. Each row reports how often BEARS strictly dominates the other methods and the mean, minimum, maximum HVIR out of 30 runs for a given release planning problem. The bottom row summarizes the results over all problems: out of 960 runs, BEARS strictly dominates NPV-*deterministic* and VP-*deterministic* in 97% and 96% of the runs, respectively. Furthermore, the BEARS shortlists have an hypervolume that is on average 18% and 17% higher than the NPV-*deterministic* and VP-*deterministic* shortlists, respectively. The hypervolume improvements range from 3% to 42%. All observed differences in hypervolumes between BEARS and the two other methods are statistically significant (Mann-Whitney U test with $p < .05$; see Appendix C).

In conclusion, the results show that:

BEARS shortlists release plans with higher expected NPV and expected punctuality than methods that ignore uncertainty.

We have also compared the method's run times. Detailed data can be found in Appendix C. On average, BEARS is 25 times slower than NPV-*deterministic* and VP-*deterministic*. For our local government example with $H = 3$, BEARS takes on average 48 seconds compared to 4 and 2 seconds for NPV-*deterministic* and VP-*deterministic*, respectively. For our largest product backlog of 200 work items and $H = 5$, BEARS takes nearly 9 minutes to complete compared to 30 and 10 seconds for NPV-*deterministic* and VP-*deterministic*. The run-time increase is due to the simulation of $10^4$ release scenarios used to reason about uncertainty. Although BEARS is significantly slower, its run time remains acceptable and does not prevent the method being used during release planning workshops. In such workshops, release plans are typically shortlisted only once after all work items effort and value estimates have been elicited.

## 5.3 Threats to Validity

*External validity.* Our results may not generalise beyond the 32 release planning problems considered in this experiment. Release planning problems where work items have different uncertainty, or where other model parameters such as team capacity have different values, or where the backlog has more or less dependencies could lead to different results. Further experiments on other problems are needed to refine the findings. There may be specific situations where ignoring uncertainty is of no consequence: for example, when all effort and value uncertainty are small, or when all work items have similar effort and value, or when the backlog dependencies are such that only a few release plans are feasible. Nevertheless, this experiment showed important differences *can* exist between BEARS and methods that ignore uncertainty, and we found important differences in all release planning problems studied so far.

The number of dependencies per work items is one of the factors that may affect the results. One limitation of our experiment is that in the synthetic product backlogs, the number of dependencies per work items is slightly lower than in the actual product backlogs: 11/30 (37%), 15/50 (30%), 30/100 (30%), and 51/200 (25%), for the synthetic product backlogs, against 23/20 (113%), 11/25 (44%) and 65/50 (130%) for the local government project, release planner, and the word processor, respectively. We ignore the RALIC project for which no dependencies had been elicited. To check whether the number of dependencies in the synthetic product backlogs may have affected the results, we have replicated the experiment by varying the number of dependencies per work items on the synthetic product backlogs of size 30 and 50. We limit this partial replication to these two product backlog sizes because they correspond to the sizes of real product backlogs reported in the literature [73]. We consider ratios of dependencies per work items of 0%, 50%, 100%, 150% and 200% that reflect and exceed the range of number of dependencies per work items in the real product backlogs. All dependencies are generated at random, with a constraint on their direction (from higher to lower work item identification number) in order to avoid circular dependencies. The results are similar to those of the original experiment: BEARS strictly dominates NPV-*deterministic* in 1192 of the 1200 runs (99%) and VP-*deterministic* in 1188 of the 1200 runs (99%). The mean HVIR of BEARS over NPV-*deterministic* and VP-*deterministic* is 1.19 in both cases. In these results, the ratio of dependencies to work items does not affect strict dominance (which is always at least 98%) and it slightly affects HVIR: the mean HVIR is between 1.16 and 1.19 when the ratio of dependencies to work items is 0%, 50% or 100%, and between 1.22 and 1.24 when the ratio is 150% or 200%. These additional results confirm the initial results that BEARS shortlist release plans with higher expected NPV and expected punctuality than methods that ignore uncertainty.

*Internal validity.* Our experiment considers that a release plan is better than another if it has higher expected net present value and higher expected punctuality. We justify this approach in Section 5.1.6 by arguing that ENPV and expected punctuality provide a better model of the release planners' true preferences than NPV or value points alone.

We have used NPV-*deterministic* and VP-*deterministic* as representatives of release planning methods that ignore uncertainty. We have also simulated the application of NPV-*deterministic* and VP-*deterministic* by deriving their effort and value estimates from the corresponding probability distributions in BEARS. This approach was designed to minimize differences between BEARS and methods that ignore uncertainty. Comparing BEARS to other methods that ignore uncertainty or using different effort and value estimates (not necessarily consistent with the BEARS estimates) would most likely amplify the differences found in our experiments.

Our experiment uses NSGA-II as MOEAs for all three methods. We used NSGA-II instead of SPEA2 and MOCell because NSGA-II is a commonly used "default" optimisation algorithm in search-based software engineering [67]. Using different MOEAs is unlikely to invalidate the conclusion that BEARS shortlists release plans with higher expected NPV and expected punctuality than methods that ignore uncertainty because the most important difference between BEARS and methods that ignore uncertainty is the optimisation model, not the optimisation algorithm. To test this claim, we have replicated our experiment for the local government project using SPEA2 and MOCell instead of NSGA-II. The results show that BEARS outperforms the other methods with all three MOEAs. When comparing BEARS to NPV-*deterministic*, the strict dominance rate and the mean HVIR are 89% and 1.30 with NSGA-II, 95% and 1.56 with SPEA, and 95% and 1.52 with MOCell. When comparing BEARS to NPV-*deterministic*, the strict dominance rate and the mean HVIR are 90% and 1.27 with NSGA-II, 98% and 1.47 with SPEA, and 87% and 1.41 with MOCell. Section 6 presents a more detailed comparison of the three MOEAs on BEARS release planning problems.

*Out of scope.* Our experiment focussed on analysing differences between optimal solutions to the mathematical optimisation models at the heart of BEARS and methods that ignore uncertainty. We have ignored other differences that are important in practice. In particular, we have not evaluated the uncertainty elicitation method (Section 4.1), the ability of release planners to understand BEARS' outputs, the overall cost of applying the method, and the general perceived benefits and limitations of the method by release planners, development teams and other stakeholders.

## 6 SECONDARY EXPERIMENTS

We have conducted two additional experiments that are secondary to the paper's main objective of evaluating the impact of analysing uncertainty on release planning but are nevertheless relevant to evaluate BEARS. This section summarizes their main findings; details can be found the online Appendices A and B.

*BEARS vs. Fixed-scope methods.* The first experiment compares BEARS to release planning methods under uncertainty that assume fixed-scope releases. For this experiment, we compared BEARS to two methods: NPV-*fixed-scope* and VP-*fixed-scope*. The first is similar to BEARS except that it assumes fixed-scope releases, the second is an extension of EVOLVE-II that allows for uncertain effort estimates [63]. The experiment design is identical to the previous one and uses the same 32 release planning problems. The results show that BEARS strictly dominates NPV-*fixed-scope* and VP-*fixed-scope* in 79% and 89% of the runs, respectively. Furthermore, BEARS's shortlists have an hypervolume that is on average 11% and 12% higher than NPV-*fixed-scope* and VP-*fixed-scope*, respectively. In conclusion: in the context of fixed-date release cycles, BEARS shortlist better release plans than methods that assume fixed-scope releases.

*Evaluating BEARS Optimisation Algorithms.* The second experiment evaluates the performance of BEARS MOEAs. For this experiment, we compared the release plans shortlisted by BEARS using three different MOEAs, NSGA-II, SPEA2 and MoCell, on the same 32 release planning problems. We also compared the MOEAs against a random search augmented with the same constraints violation detection and repair technique used by the MOEAs. The results show that the three MOEAs perform better than the random search and that the differences are statistically significant. The results show no statistically significant differences between SPEA2 and MOCell, and a slight advantage of these two algorithms over NSGA-II on our 32 release planning problems. The results therefore suggest that SPEA2 and MOCell may find slightly better shortlists than NSGA-II for BEARS optimisation problems. The purpose of this experiment was merely to check that these three MOEAs perform reasonably well on BEARS model. We expect that future research will develop improved algorithms able to find better shortlists with fewer evaluations.

## 7 BEARS LIMITATIONS AND FUTURE WORKS

BEARS was developed primarily to support our experiments. For practical use, the method and tool have a number of limitations to be addressed in future work.

*Applicability and cost.* As mentioned earlier, we have not yet evaluated the method in an industrial context. As a result, the applicability and cost of BEARS in industrial contexts are unknown. Other release planning methods under uncertainty suffer from the same limitation. In contrast, EVOLVE-II and other methods that ignore uncertainty have been applied in industry and their benefits well documented [4, 45, 51]. We hope the results of our experiments can serve to justify future industry trials of release planning methods under uncertainty.

*Modelling assumptions.* The economic model used in BEARS (Eq. 7 to 10) relies on simplifying assumptions that may affect the accuracy of their net present value and punctuality estimations. Notably, the model assumes that work items are independent. In practice, the values of two work items may depend on a third variable making this assumption invalid (e.g. the values of online services related to council tax payments all depend on the number of residents who pay council taxes). This limitation can be addressed by developing more complex models but this would make the method more difficult to apply because it will require release planners to develop project-specific models of business value. As always, the right balance needs to be found between model complexity and accuracy. We believe that BEARS is a good compromise: it retains much of the simplicity of methods that ignore uncertainty (from a user perspective, only the effort and value estimation step has changed) while providing important improvement in shortlisted release plans.

*Tool limitations.* The tool supporting BEARS is a prototype developed primarily to conduct our experiments. The tool can evaluate and shortlist release plans (Sections 4.2 to 4.3) but it currently relies on an external tool for uncertainty elicitation (Section 4.1) and does not include features to help release planners select their preferred plan among the shortlisted ones. Such features are essential for future industrial applications.

## 8 RELATED WORK

*Release planning methods.* Section 3 already provided a detailed guided tour of release planning methods. We have seen that most methods ignore uncertainty [5, 68] and that the few that analyse uncertainty [10, 46, 49, 58, 63] rely on a fixed-scope assumption that is inconsistent with current industrial practices [45]. In contrast, BEARS analyses uncertainty in the context of fixed-date release cycles, the release process most commonly used in industry [45]. The technical novelties in BEARS are (i) a novel model for simulating fixed-date release plans under uncertainty (Section4.2); (ii) a novel metric for measuring the expected punctuality of release plans (Section 4.2.3); and (iii) the first automated tool to support release planning for fixed-date release cycles (Section 4.3).

*Empirical studies.* The paper's main contribution is the experiment comparing release planning with and without analysing uncertainty. Previous empirical studies of release planning methods have either focussed on evaluating release planning methods in industry [4, 45, 51] or on comparing the performance of alternative optimisation algorithms (as we do in Appendix B). A recent paper surveys 38 different evaluations of MOEAs for software release planning and presents the most comprehensive such evaluation to date [73]. Such evaluations compare the outputs of different

MOEAs in the context of a single release planning method. In contrast, we present the first empirical study that compares different release planning methods against each other.

*Strategic vs. operational release planning.* In this paper, we studied methods for *strategic release planning*, the activity of assigning features to future releases. In contrast, *operational release planning* is the activity of assigning tasks to developers within a single iteration [1, 2]. Sprint planning in Scrum is an example of operational release planning [60]. Operational release planning is performed more frequently than strategic release planning and deals with much finer grained work items (e.g. user stories and tasks) than considered during strategic release planning. The release plan generated by strategic release planning is an input to subsequent operational release planning. A method to analyse uncertainty during operational release planning has been proposed in previous work [1]. As in BEARS, the method uses Monte-Carlo simulations to analyse uncertainty. In this context, the Monte-Carlo simulation is applied to a process simulation model of operational release plans, whereas in BEARS it is applied to a cost-value model of strategic release plans. The method to evaluate uncertainty in operational release plans performs a probabilistic sensitivity analysis to evaluate the impact of variations in the values of model parameters such as developers' productivity and the number and sizes of feature to be developed, on the total time required to develop a given set of tasks. Unlike BEARS, the proposed method does not use any optimisation method to shortlist optimal release plans. The paper also has no experiment comparing operational release planning methods that either analyse or ignore uncertainty. In future work, the research approach presented in this paper could be adapted to the context of operational release planning.

*Requirements and architecture decisions under uncertainty.* The Bayesian approach used in BEARS is similar to one used to support requirements and architecture decisions under uncertainty [13, 39]. The decision models supporting such requirements and architecture decisions are based on quantitative goal models [28, 40, 70]. They allow a richer, more detailed modelling of stakeholders' goals and business value than BEARS, but they do not support reasoning about multiple iterations as done in BEARS. BEARS introduces novel simulation and optimisation models that are specific to release planning.

## 9 CONCLUSION

The paper has shown the importance of analysing uncertainty during release planning. Analysing uncertainty generates release plans that have higher expected net present value and expected punctuality than when uncertainty is ignored. Analysing uncertainty also give release planners important information about release plans' expected punctuality that remain hidden when uncertainty is ignored. Such information enables informed discussion about finding the right trade-off between release plans that are too ambitious (high expected net present value but low expected punctuality) and too conservative (high expected punctuality but low expected net present value).

BEARS is the first release planning method under uncertainty designed for the industry practice of fixed-date release cycles. We have also presented the first experiment comparing a release planning method that analyse uncertainty to release planning methods that ignore uncertainty. Having shown the potential benefits of analysing uncertainty, we hope that our experiment will motivate further research in this area, notably industry trials of release planning under uncertainty and intensified research about uncertainty elicitation and analysis in software effort and value estimates (e.g. [13, 21, 27, 34, 39, 66]). In future work, we also plan on extending BEARS to support a wider range of activities such as: (i) analysing values from multiple perspectives and taking fairness into consideration [23]; (ii) managing technical debt during release planning [6, 65]; (iii) supporting decisions about what experiments to perform (e.g. as A/B tests) and what data to collect to improve release planning decisions; and (iv) analysing objective data (e.g. from the software development process, software usage, and users' feedback) to update uncertainty about feature's effort and value so as to inform future decisions [47].

## REFERENCES

[1] Ahmed Al-Emran, Puneet Kapur, Dietmar Pfahl, and Günther Ruhe. [n.d.]. Studying the impact of uncertainty in operational release planning - An integrated method and its initial evaluation. 52, 4 ([n. d.]), 446–461.

[2] Ahmed Al-Emran and Dietmar Pfahl. [n.d.]. Operational Planning, Re-planning and Risk Analysis for Software Releases. In *8th International Conference on Product-Focused Software Process Improvement, PROFES 2007, Riga, Latvia, July 2-4, 2007.* (2007) *(Lecture Notes in Computer Science)*, Vol. 4589. Springer, 315–329.

[3] Antonio J Alencar, Carlos AS Franco, Eber A Schmitz, and Alexandre L Correa. 2014. A statistical approach for the maximization of the financial benefits yielded by a large set of MMFs and AEs. *Computing and Informatics* 32, 6 (2014), 1147–1169.

[4] Amandeep, Günther Ruhe, and Mark Stanford. 2004. Intelligent Support for Software Release Planning. In *PROFES*. 248–262.

[5] David Ameller, Carles Farré, Xavier Franch, and Guillem Rufian. 2016. A Survey on Software Release Planning Models. In *PROFES*. 48–65.

[6] Areti Ampatzoglou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Paris Avgeriou. 2015. The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology* 64 (2015), 52–73.

[7] Allysson Allex Araújo, Matheus Paixao, Italo Yeltsin, Altino Dantas, and Jerffeson Souza. 2017. An architecture based on interactive optimization and machine learning applied to the next release problem. *Automated Software Engineering* 24, 3 (2017), 623–671.

[8] Andrea Arcuri and Lionel Briand. 2011. A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In *ICSE*. 10.

[9] Anthony J. Bagnall, Victor J. Rayward-Smith, and Ian M Whittley. 2001. The next release problem. *Information and software technology* 43, 14 (2001), 883–890.

[10] Breno Peixoto Barbosa, Eber Assis Schmitz, and Antonio Juarez Alencar. 2008. Generating software-project investment policies in an uncertain environment. In *Systems and Information Engineering Design Symposium (SIEDS 2008)*. IEEE, 178–183.

[11] Stefan Biffl, Aybuke Aurum, Barry Boehm, Hakan Erdogmus, and Paul Grünbacher. 2006. *Value-based software engineering*. Springer.

[12] Barry W. Boehm. 1988. A spiral model of software development and enhancement. *Computer* 21, 5 (1988), 61–72.

[13] Saheed A Busari and Emmanuel Letier. 2017. RADAR: A lightweight tool for requirements and architecture decision analysis. In *ICSE*. 552–562.

[14] Murray Cantor. 2011. Calculating and Improving ROI in Software and System Programs. *Commun. ACM* 54, 9 (2011), 10.

[15] Mike Cohn. 2005. *Agile estimating and planning*. Pearson Education.

[16] Francisco Gomes de Oliveira Neto, Richard Torkar, Robert Feldt, Lucas Gren, Carlo A Furia, and Ziwei Huang. 2019. Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. *Journal of Systems and Software* 156 (2019), 246–267.

[17] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[18] Tom DeMarco and Timothy Lister. 2003. *Waltzing with Bears: Managing Risk on Software Projects.* Dorset House Publishing.

[19] Mark Denne and Jane Cleland-Huang. 2003. *Software by numbers: Low-risk, high-return development.* Prentice Hall.

[20] Mark Denne and Jane Cleland-Huang. 2004. The incremental funding method: Data-driven software development. *IEEE software* 21, 3 (2004), 39–47.

[21] Hakan Erdogmus, John Favaro, and Michael Halling. 2006. Valuation of Software Initiatives Under Uncertainty: Concepts, Issues, and Techniques. In *Value-Based Software Engineering*. Springer Berlin Heidelberg, 39–66.

[22] Filomena Ferrucci, Mark Harman, Jian Ren, and Federica Sarro. 2013. Not going to take this anymore: multi-objective overtime planning for software engineering projects. In *35th International Conference on Software Engineering (ICSE)*. IEEE.

[23] Anthony Finkelstein, Mark Harman, S Afshin Mansouri, Jian Ren, and Yuanyuan Zhang. 2008. Fairness Analysis in Requirements Assignments. In *International Requirements Engineering Conference*. 115–124.

[24] Bent Flyvbjerg and Alexander Budzier. 2011. Why your IT project may be riskier than you think. *Harvard Business Review* 89, 9 (2011).

[25] Sander Greenland, Stephen J Senn, Kenneth J Rothman, John B Carlin, Charles Poole, Steven N Goodman, and Douglas G Altman. 2016. Statistical tests, P values, confidence intervals, and power: a guide to misinterpretations. *European journal of epidemiology* 31, 4 (2016), 337–350.

[26] Des Greer and Guenther Ruhe. 2004. Software release planning: an evolutionary and iterative approach. *Information and software technology* 46, 4 (2004), 243–253.

[27] T. Gruschke and M. Jorgensen. 2005. Assessing Uncertainty of Software Development Effort Estimates: The Learning from Outcome Feedback. In *11th IEEE International Software Metrics Symposium (METRICS'05)*. IEEE.

[28] W. Heaven and E. Letier. 2011. Simulating and optimising design decisions in quantitative goal models. In *International Requirements Engineering Conference*. 79–88.

[29] D.W. Hubbard. 2010. *How to Measure Anything: Finding the Value of Intangibles in Business*. Wiley.

[30] Douglas W Hubbard and Richard Seiersen. 2016. *How to measure anything in cybersecurity risk*. Wiley.

[31] Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. 2015. Modified distance calculation in generational distance and inverted generational distance. In *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 110–125.

[32] Magne Jørgensen. 2014. What we do and don't know about software development effort estimation. *IEEE software* 31, 2 (2014), 37–40.

[33] Magne Jorgensen and Martin Shepperd. 2007. A systematic review of software development cost estimation studies. *IEEE Transactions on software engineering* 33, 1 (2007).

[34] Magne Jørgensen, Karl Halvor Teigen, and Kjetil Moløkken. 2004. Better sure than safe? Over-confidence in judgement based software development effort prediction intervals. *Journal of Systems and Software* 70, 1-2 (02 2004), 79–93.

[35] Muhammad Rezaul Karim and Guenther Ruhe. 2014. Bi-objective genetic search for release planning in support of themes. In *International Symposium on Search Based Software Engineering*. 123–137.

[36] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on software engineering* 28, 8 (2002), 721–734.

[37] Craig Larman and Victor R Basili. 2003. Iterative and incremental developments. a brief history. *Computer* 36, 6 (2003), 47–56.

[38] Dean Leffingwell. 2010. *Agile software requirements: lean requirements practices for teams, programs, and the enterprise.* Addison-Wesley Professional.

[39] Emmanuel Letier, David Stefan, and Earl T. Barr. 2014. Uncertainty, Risk, and Information Value in Software Requirements and Architecture. In *ICSE* (Hyderabad, India). 883–894.

[40] E. Letier and A. van Lamsweerde. 2004. Reasoning about partial goal satisfaction for requirements and design engineering. In *FSE*. 53–62.

[41] Lingbo Li, Mark Harman, Emmanuel Letier, and Yuanyuan Zhang. 2014. Robust next release problem: handling uncertainty during optimization. In *GECCO*. 1247–1254.

[42] Lingbo Li, Mark Harman, Fan Wu, and Yuanyuan Zhang. 2016. The value of exact analysis in requirements selection. *IEEE Transactions on Software Engineering* (2016), 1–1.

[43] Miqing Li, Tao Chen, and Xin Yao. 2018. A critical review of: a practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering: essay on quality indicator selection for SBSE. In *ICSE: NIER*.

[44] Soo Ling Lim and Anthony Finkelstein. 2012. StakeRare: using social networks and collaborative filtering for large-scale requirements elicitation. *IEEE transactions on software engineering* 38, 3 (2012).

[45] Markus Lindgren, Rikard Land, Christer Norström, and Anders Wall. 2008. Key aspects of software release planning in industry. In *19th Australian Conference on Software Engineering*. 320–329.

[46] Kevin Logue and Kevin McDaid. 2008. Agile release planning: Dealing with uncertainty in development time and business value. In *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2008)*. IEEE, 437–442.

[47] Walid Maalej, Maleknaz Nayebi, Timo Johann, and Guenther Ruhe. 2016. Toward data-driven requirements engineering. *IEEE Software* 33, 1 (2016), 48–54.

[48] Eduardo Mattos, Marcelo Vieira, Eber Assis Schmitz, and Antonio Juarez Alencar. 2014. Applying game theory to the incremental funding method in software projects. *Journal of Software* 9, 6 (2014), 14–35.

[49] Kevin McDaid, Des Greer, Frank Keenan, Paul Prior, Gerry Coleman, and Philip S Taylor. 2006. Managing Uncertainty in Agile Release Planning.. In *SEKE*. 138–143.

[50] Blakeley B McShane, David Gal, Andrew Gelman, Christian Robert, and Jennifer L Tackett. 2019. Abandon statistical significance. *The American Statistician* 73, sup1 (2019), 235–245.

[51] Joseph Momoh and Guenther Ruhe. 2006. Release planning process improvement — an industrial case study. *Software Process: Improvement and Practice* 11, 3 (2006), 295–307.

[52] David E Morris, Jeremy E Oakley, and John A Crowe. 2014. A web-based tool for eliciting probability distributions from experts. *Environmental modelling & software* 52 (2014), 1–4.
[53] National Audit Office. 2013. *Over-optimism in government projects.* http://www.nao.org.uk/report/optimism-bias-paper (Last Accessed: August 2021).
[54] Antonio J Nebro, Juan J Durillo, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. 2009. Mocell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems* 24, 7 (2009), 726–746.
[55] Antonio J. Nebro, Juan J. Durillo, and Matthieu Vergne. 2015. Redesigning the jMetal Multi-Objective Optimization Framework. In *GECCO* (Madrid, Spain). New York, NY, USA, 1093–1100.
[56] A. O'Hagan, C.E. Buck, A. Daneshkhah, J.R. Eiser, P.H. Garthwaite, D.J. Jenkinson, J.E. Oakley, and T. Rakow. 2006. *Uncertain Judgements: Eliciting Experts' Probabilities.* Wiley.
[57] Anthony O'Hagan and Jeremy E Oakley. 2004. Probability is perfect, but we can't elicit it perfectly. *Reliability Engineering & System Safety* 85, 1-3 (2004), 239–248.
[58] Olawole Oni and Emmanuel Letier. 2016. Optimizing the Incremental Delivery of Software Features Under Uncertainty. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ).* 36–41.
[59] Antônio Mauricio Pitangueira, Paolo Tonella, Angelo Susi, Rita Suzana Pitangueira Maciel, and M Barros. 2017. Minimizing the stakeholder dissatisfaction risk in requirement selection for next release planning. *Information and Software Technology* 87 (2017), 104–118.
[60] Kenneth S Rubin. 2012. *Essential Scrum: A practical guide to the most popular Agile process.* Addison-Wesley.
[61] Günther Ruhe. 2010. *Product Release Planning: Methods, Tools and Applications.* CRC Press.
[62] Guenther Ruhe et al. 2008. A systematic approach for solving the wicked problem of software release planning. *Soft Computing* 12, 1 (2008), 95–108.
[63] Günther Ruhe and Des Greer. 2003. Quantitative Studies in Software Release Planning Under Risk and Resource Constraints. In *International Symposium on Empirical Software Engineering (ISESE '03).*
[64] José Sagrado, Isabel M. Águila, and Francisco J. Orellana. 2015. Multi-objective Ant Colony Optimization for Requirements Selection. *Empirical Software Engineering* 20, 3 (2015), 577–610.
[65] R. S. Sangwan, A. Negahban, R. L. Nord, and I. Ozkaya. 2020. Optimization of Software Release Planning Considering Architectural Dependencies, Cost, and Value. *IEEE Transactions on Software Engineering* (2020).
[66] Federica Sarro, Alessio Petrozziello, and Mark Harman. 2016. Multi-objective software effort estimation. In *38th International Conference on Software Engineering (ICSE).* ACM Press.
[67] Abdel Salam Sayyad and Hany Ammar. 2013. Pareto-optimal search-based software engineering (POSBSE): A literature survey. In *2013 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE).* IEEE, 21–27.
[68] Mikael Svahnberg, Tony Gorschek, Robert Feldt, Richard Torkar, Saad Bin Saleem, and Muhammad Usman Shafique. 2010. A systematic review on strategic release planning models. *Information and software technology* 52, 3 (2010).
[69] Steve Tockey. 2005. *Return on software.* Addison-Wesley.
[70] A. van Lamsweerde. 2009. *Requirements engineering: from system goals to UML models to software specifications.* Wiley.
[71] Yuanyuan Zhang, Enrique Alba, Juan J Durillo, Sigrid Eldh, and Mark Harman. 2010. Today/future importance analysis. In *GECCO.* ACM, 1357–1364.
[72] Yuanyuan Zhang, Mark Harman, and Soo Ling Lim. 2013. Empirical evaluation of search based requirements interaction management. *Information and Software Technology* 55, 1 (2013), 126–152.
[73] Yuanyuan Zhang, Mark Harman, Gabriela Ochoa, Guenther Ruhe, and Sjaak Brinkkemper. 2018. An Empirical Study of Meta-and Hyper-Heuristic Search for Multi-Objective Release Planning. *TOSEM* 27, 1 (2018), 3.
[74] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. 2001. SPEA2: Improving the strength Pareto evolutionary algorithm. (2001).
[75] Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE transactions on Evolutionary Computation* 3, 4 (1999), 257–271.
[76] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. 2003. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation* 7, 2 (2003), 117–132.

## A COMPARING BEARS TO FIXED-SCOPE METHODS

This appendix reports an experiment comparing BEARS to release planning methods under uncertainty that assume fixed-scope releases. The experiment's objective is to study whether assuming fixed-scope releases is harmful when applied to projects with fixed-date release cycles.

### A.1 Experiment Design

The research question is:

> In the context of fixed-date release cycles, does BEARS shortlist better release plans than methods that assume fixed-scope releases?

The experiment design is identical to the one used to study differences between BEARS and methods that ignore uncertainty. In particular, we use the same release planning problems, evaluation metrics and experimental set-up. The only difference is that BEARS is now compared to two release planning methods under uncertainty that assume fixed-scope releases: NPV-*fixed-scope* and VP-*fixed-scope*. These methods are summarized in Table 5.

NPV-*fixed-scope* is a variant of BEARS that uses the same uncertain effort and value estimates as BEARS but simulates effort uncertainty assuming fixed-scope releases instead of flexible-scope. We have described simulation methods for both cases in Section 4.2. Comparing BEARS to NPV-*fixed-scope* allows us to study the difference between fixed-scope and flexible-scope simulations independently from other differences.

VP-*fixed-scope* is based on a previously published variant of EVOLVE that models uncertainty about effort but not value [63]. The method simulates effort uncertainty assuming fixed-scope releases. The optimisation problem has two objectives: maximise value points (Eq. 3) and maximise the probability of delivering all features on time. The latter is defined as:

$$P(\forall i \in [1..H] : \sum_{\{w \in WI \mid p(w)=i\}} \mathit{effort}(w) \leq \mathit{capacity}(i))$$

The shortlists in NPV-*fixed-scope* and VP-*fixed-scope* are the set of Pareto-optimal solutions returned by the MOEA. Shortlists of different methods can therefore be of different sizes.

Both methods use the same MOEAs with repair as BEARS. For consistency, we have tested all methods using NSGA-II as MOEA. This reduces the risk that differences between the methods' shortlists are due to the MOEA rather than to differences in the optimisation models.

### A.2 Results

Table 6 shows the results. Each row shows the result for a specific release planning problem. The bottom row summarises the results: out of 960 runs, the BEARS shortlist strictly dominates the NPV-*fixed-scope* and VP-*fixed-scope* shortlists in 79% and 89% of the runs, respectively. Furthermore, the BEARS shortlist has an hypervolume that is on average 11% and 12% higher than the NPV-*fixed-scope* and VP-*fixed-scope* shortlists, respectively. The hypervolume improvements range from 1% to 32%. All observed differences in hypervolumes between BEARS and the two other methods are statistically significant (Mann-Whitney U test with $p < .05$, see details in Table 9).

In the context of fixed-date release cycles, BEARS shortlist release plans with higher expected NPV and expected punctuality than methods that assume fixed-scope releases. In our experiment, the hypervolume improvement ranges from 1% to 32%, with an average of 11%.

Table 5. Release planning methods in experiment comparing BEARS to methods that assume fixed-scope releases

| Model | Effort estimates | Value estimates | Simulation Method | Optimisation problem |
|---|---|---|---|---|
| BEARS | uncertain person-days | uncertain economic value | stochastic, flexible-scope | Maximise expected NPV; Maximise expected punctuality |
| NPV-*fixed-scope* | uncertain person-days | uncertain economic value | stochastic, fixed-scope | Maximise expected NPV; Maximise on-time probability |
| VP-*fixed-scope* | uncertain story points | value points | stochastic, fixed-scope | Maximise value points; Maximise on-time probability |

Table 6. Strict Dominance and Hypervolume Improvement Ratio of BEARS over NPV-*fixed-scope* and VP-*fixed-scope*.

| | | BEARS vs. NPV-*fixed-scope* | | | BEARS vs. VP-*fixed-scope* | | |
|---|---|---|---|---|---|---|---|
| Product Backlog | H | Strict Dominance | HVIR Mean | HVIR Range | Strict Dominance | HVIR Mean | HVIR Range |
| Local Government Project | 2 | 18/30 | 1.05 | 1.00 –1.12 | 18/30 | 1.06 | 1.03 –1.07 |
| | 3 | 21/30 | 1.12 | 1.05 –1.33 | 19/30 | 1.12 | 1.07 –1.23 |
| | 4 | 23/30 | 1.15 | 1.10 –1.27 | 22/30 | 1.11 | 1.07 –1.17 |
| | 5 | 27/30 | 1.16 | 1.11 –1.24 | 21/30 | 1.11 | 1.05 –1.19 |
| Release Planner | 2 | 21/30 | 1.01 | 0.93 –1.06 | 25/30 | 1.02 | 0.95 –1.07 |
| | 3 | 28/30 | 1.05 | 1.01 –1.08 | 30/30 | 1.06 | 1.03 –1.09 |
| | 4 | 30/30 | 1.06 | 1.03 –1.11 | 30/30 | 1.05 | 1.03 –1.08 |
| | 5 | 30/30 | 1.08 | 1.05 –1.12 | 30/30 | 1.06 | 1.03 –1.08 |
| Word Processor | 2 | 26/30 | 1.01 | 0.94 –1.03 | 20/30 | 1.00 | 0.94 –1.05 |
| | 3 | 23/30 | 1.03 | 1.00 –1.05 | 27/30 | 1.03 | 1.00 –1.05 |
| | 4 | 30/30 | 1.03 | 1.02 –1.04 | 30/30 | 1.05 | 1.04 –1.05 |
| | 5 | 30/30 | 1.04 | 1.01 –1.07 | 26/30 | 1.02 | 0.99 –1.03 |
| RALIC | 2 | 30/30 | 1.08 | 1.04 –1.14 | 30/30 | 1.07 | 1.03 –1.12 |
| | 3 | 29/30 | 1.08 | 1.02 –1.15 | 27/30 | 1.06 | 0.98 –1.12 |
| | 4 | 29/30 | 1.08 | 1.03 –1.15 | 30/30 | 1.07 | 1.01 –1.11 |
| | 5 | 27/30 | 1.08 | 1.02 –1.17 | 20/30 | 1.06 | 1.00 –1.09 |
| Synthetic-30 | 2 | 20/30 | 1.02 | 1.00 –1.05 | 18/30 | 1.03 | 1.01 –1.09 |
| | 3 | 30/30 | 1.08 | 1.04 –1.13 | 20/30 | 1.08 | 1.03 –1.11 |
| | 4 | 30/30 | 1.12 | 1.06 –1.17 | 21/30 | 1.11 | 1.06 –1.15 |
| | 5 | 30/30 | 1.15 | 1.11 –1.23 | 27/30 | 1.14 | 1.11 –1.18 |
| Synthetic-50 | 2 | 20/30 | 1.02 | 1.00 –1.06 | 20/30 | 1.05 | 1.04 –1.08 |
| | 3 | 21/30 | 1.08 | 1.03 –1.12 | 20/30 | 1.08 | 1.05 –1.12 |
| | 4 | 24/30 | 1.14 | 1.08 –1.20 | 18/30 | 1.13 | 1.05 –1.25 |
| | 5 | 28/30 | 1.20 | 1.14 –1.27 | 23/30 | 1.17 | 1.10 –1.26 |
| Synthetic-100 | 2 | 28/30 | 1.09 | 1.04 –1.13 | 21/30 | 1.07 | 1.04 –1.12 |
| | 3 | 29/30 | 1.16 | 1.10 –1.26 | 24/40 | 1.15 | 1.09 –1.22 |
| | 4 | 28/30 | 1.22 | 1.12 –1.32 | 20/30 | 1.20 | 1.09 –1.32 |
| | 5 | 29/30 | 1.26 | 1.18 –1.35 | 20/30 | 1.25 | 1.16 –1.34 |
| Synthetic-200 | 2 | 27/30 | 1.16 | 1.09 –1.25 | 20/30 | 1.15 | 1.07 –1.23 |
| | 3 | 30/30 | 1.24 | 1.13 –1.31 | 22/30 | 1.21 | 1.12 –1.29 |
| | 4 | 29/30 | 1.33 | 1.17 –1.55 | 20/30 | 1.32 | 1.17 –1.92 |
| | 5 | 30/30 | 1.36 | 1.18 –1.54 | 25/30 | 1.37 | 1.25 –1.69 |
| **Overall** | | 89% (855/960) | 1.12 | 1.01 –1.32 | 79% (754/960) | 1.11 | 1.01 –1.29 |

In terms of run-times, we measured BEARS to be between 2 to 8 times slower than the fixed-scope methods (see Table 10). For example, for our local government project and a planning horizon of 3, the average run-time for BEARS is 48 seconds, compared to 20 and 12 seconds for NPV-*fixed-scope* and VP-*fixed-scope*, respectively. For our largest problem with 200 work items and a planning horizon of 5, BEARS average run-time is nearly 9 minutes compared to 2 minutes 30 and 2 minutes for NPV-*fixed-scope* and VP-*fixed-scope*. The run-time difference is due to the additional complexity of simulating release plans with flexible scope over fixed-scope (see Section 4.2).

### A.3 Threats to Validity

*External validity.* As for the main experiment, the results of this experiment may not generalise beyond the 32 release planning problems considered in this study. The results nevertheless show that differences do exist between BEARS and methods that assume fixed-scope releases and justify using BEARS instead of fixed-scope methods in the context of fixed-date release cycles.

*Internal validity.* We have shown that BEARS is better than methods that assume fixed-scope releases in the context of fixed-date release cycles for which BEARS was designed. As in the main experiment, we assumed that BEARS optimisation criteria are consistent with the release planner's true preferences. Naturally, fixed-scope methods would be more appropriate than BEARS in the context of a project with fixed-scope releases.

## B EVALUATING BEARS OPTIMISATION ALGORITHMS

This experiment evaluates the performance of BEARS MOEAs. We aim to answer the following questions:

(RQ1) Do the three MOEAs used in BEARS perform better than a random search?
(RQ2) Do the three MOEAs used in BEARS have important differences in performance?

Our objective is not to propose novel MOEAs, nor to look for the best possible algorithms for solving BEARS optimisation problems. This experiment merely aims to check that existing MOEAs are suitable for solving BEARS optimisation problems. Developing and evaluating more efficient algorithms for solving BEARS optimisation problems is left as future work.

### B.1 Experiment Design

The three MOEAs in our implementation of BEARS are NSGA-II, SPEA2, and MOCell. We selected these MOEAs because they have readily available implementations in JMetal [55], the optimisation framework used in our tool. We compare these algorithms against a random search augmented with the same constraints violation detection and repair technique used with the MOEAs (see Section 4.3). The random search is configured to evaluate the same number of valid release plans as the MOEAs $(25,000)$.

We have executed each of the 4 algorithms 30 times on the same 32 release planning problems used in our main experiment. This makes up a total of $3,840$ runs. For each run, we retrieve the generated shortlist and measure its Hypervolume (HV) [75] and modified Inverted Generational Distance (IGD+) [31]. In this experiment, we compute HV using *normalised* expected NPV values for each problem, where the minimum is 0 and maximum is the highest expected NPV found in all evaluated release plans for that problem. The HV for all problems are thus all measured on the same scale. Better shortlists have *higher* HV. The IGD+ of a solution set $A$ is the average distance between the true Pareto-optimal solutions and the region in the objective space dominated by $A$ [31]. When, as in our case, the true Pareto optimal solutions are unknown, these solutions are approximated by so-called *reference* Pareto-optimal solutions, which are the non-dominated solutions in the union of all solutions explored by all algorithms over all of independent runs. Better shortlists have *lower* IGD+.

We have chosen HV and IGD+ as our evaluation metrics based on guidance from Li et al. [43] that recommend these metrics in situations, like ours, where the decision makers' preferences about the qualities of solutions sets are unknown These metrics are recommended because they are compatible with the strict dominance relation and they cover all typical qualities desired of solutions sets, i.e. their convergence (how close the solutions set is to the true Pareto front), diversity (the extent to which the solutions set includes diverse solutions), and cardinality (the number of solutions in the set).

Table 7. Performance of a random search and the 3 Multi-Objective Optimisation Algorithms used in BEARS. In each row, the worst result is highlighted in dark grey and the best in light grey.

| | | Mean Hypervolume | | | | Mean IGD+ | | | |
|---|---|---|---|---|---|---|---|---|---|
| Product Backlog | H | NSGA-II | SPEA2 | MOCell | Random | NSGA-II | SPEA2 | MOCell | Random |
| Local Government Project | 2 | 0.956 | 0.947 | 0.903 | 0.825 | 0.019 | 0.011 | 0.009 | 0.080 |
| | 3 | 0.982 | 0.982 | 0.991 | 0.878 | 0.076 | 0.042 | 0.002 | 0.115 |
| | 4 | 0.973 | 0.988 | 0.968 | 0.943 | 0.021 | 0.004 | 0.002 | 0.110 |
| | 5 | 0.975 | 0.982 | 0.996 | 0.888 | 0.019 | 0.009 | 0.005 | 0.114 |
| Release Planner | 2 | 0.806 | 0.807 | 0.851 | 0.752 | 0.268 | 0.025 | 0.019 | 0.444 |
| | 3 | 0.873 | 0.949 | 0.879 | 0.820 | 0.424 | 0.088 | 0.020 | 0.635 |
| | 4 | 0.948 | 0.973 | 0.900 | 0.917 | 0.178 | 0.014 | 0.005 | 0.301 |
| | 5 | 0.945 | 0.927 | 0.969 | 0.739 | 0.047 | 0.026 | 0.001 | 0.180 |
| Word Processor | 2 | 0.938 | 0.946 | 0.949 | 0.820 | 0.560 | 0.138 | 0.032 | 1.880 |
| | 3 | 0.971 | 0.988 | 0.999 | 0.907 | 0.257 | 0.016 | 0.005 | 1.050 |
| | 4 | 0.971 | 0.986 | 0.994 | 0.915 | 0.076 | 0.035 | 0.001 | 0.525 |
| | 5 | 0.925 | 0.938 | 0.996 | 0.925 | 0.052 | 0.006 | 0.005 | 0.361 |
| RALIC | 2 | 0.984 | 0.986 | 0.996 | 0.780 | 0.038 | 0.001 | 0.002 | 6.040 |
| | 3 | 0.967 | 0.992 | 0.994 | 0.686 | 0.319 | 0.002 | 0.081 | 9.700 |
| | 4 | 0.927 | 0.971 | 0.997 | 0.669 | 0.455 | 0.005 | 0.008 | 6.260 |
| | 5 | 0.922 | 0.984 | 0.942 | 0.660 | 0.466 | 0.019 | 0.001 | 4.320 |
| Synthetic-30 | 2 | 0.594 | 0.598 | 0.517 | 0.526 | 0.579 | 0.509 | 0.025 | 0.726 |
| | 3 | 0.720 | 0.723 | 0.721 | 0.670 | 0.383 | 0.251 | 0.378 | 0.412 |
| | 4 | 0.846 | 0.947 | 0.845 | 0.818 | 0.284 | 0.008 | 0.039 | 0.353 |
| | 5 | 0.933 | 0.945 | 0.955 | 0.906 | 0.021 | 0.004 | 0.01 | 0.125 |
| Synthetic-50 | 2 | 0.718 | 0.955 | 0.714 | 0.688 | 0.216 | 0.011 | 0.161 | 0.216 |
| | 3 | 0.943 | 0.945 | 0.970 | 0.808 | 0.123 | 0.011 | 0.014 | 0.137 |
| | 4 | 0.958 | 0.962 | 0.889 | 0.914 | 0.030 | 0.009 | 0.007 | 0.161 |
| | 5 | 0.963 | 0.968 | 0.965 | 0.841 | 0.034 | 0.008 | 0.012 | 0.332 |
| Synthetic-100 | 2 | 0.886 | 0.886 | 0.687 | 0.711 | 0.136 | 0.058 | 0.134 | 0.137 |
| | 3 | 0.936 | 0.936 | 0.888 | 0.931 | 0.037 | 0.016 | 0.012 | 0.104 |
| | 4 | 0.948 | 0.948 | 0.948 | 0.758 | 0.083 | 0.014 | 0.039 | 0.265 |
| | 5 | 0.922 | 0.922 | 0.959 | 0.839 | 0.083 | 0.025 | 0.024 | 0.506 |
| Synthetic-200 | 2 | 0.726 | 0.709 | 0.727 | 0.681 | 0.023 | 0.021 | 0.021 | 0.024 |
| | 3 | 0.975 | 0.945 | 0.961 | 0.956 | 0.024 | 0.008 | 0.020 | 0.081 |
| | 4 | 0.970 | 0.911 | 0.953 | 0.887 | 0.061 | 0.003 | 0.048 | 0.274 |
| | 5 | 0.970 | 0.964 | 0.938 | 0.819 | 0.079 | 0.009 | 0.047 | 0.606 |
| **Overall** | | 0.726 | 0.829 | 0.784 | 0.670 | 0.020 | 0.003 | 0.002 | 0.110 |

## B.2 Results

Table 7 reports the mean HV and IGD+ of each algorithm over 30 runs for each release planning problem. In each row, the best result is highlighted in light grey and the worst in dark grey. We have used Mann-Whitney U test to evaluate whether the observed differences in HV and IGD+ between algorithms are statistically significant. Table 8 reports the results of these tests.

*RQ1: Do the three MOEAs used in BEARS perform better than a random search?* The results show that the three MOEAs perform better than random search and that the differences in HV and IGD+ are statistically significant. In terms of IGD+, the three MOEAS outperform random search in all 32 problems. In terms of HV, the three MOEAS outperform random search in 27 of the 32 problems. In the remaining 5 problems, random search performs better than MOCEll in 4 cases and better than SPEA2 in 1 case. Note that the random search in BEARS uses the same constraint violation detection and repair technique used by the MOEAs. Random search here is therefore more than a simple blind search. This may explain why in a few cases it performs better than one of the three MOEAs for one of the evaluation criteria, even if overall the MOEAs have better IGD+ and HV.

*RQ2: Do the three MOEAs used in BEARS have important differences in performance?* The results show no statistically significant differences between SPEA2 and MOCell, and a slight advantage of these two algorithms over NSGA-II. The experiment therefore suggest that SPEA2 and MOCell may find slightly better shortlists than NSGA-II for BEARS optimisation problems.

### B.3 Threats to Validity

Our experiment follows common guidelines and practices for evaluating the performance of stochastic multi-objective algorithms [8, 43, 73]. The most important threat to validity is the extent to which our results can be generalized beyond the 32 release planning problems. We have controlled threats to internal validity by justifying our selection of evaluation metrics and executing each algorithm 30 times on each problem, as described in Section B.1.

Table 8. Statistical Significance (p-value) of the differences between MOEAs on BEARS using Mann-Whitney U test. Highlighted cells are those where the differences are not satisfically significant (p>.05)

| | | | Random vs. | | | NSGA-II vs. | | SPEA2 vs. |
|---|---|---|---|---|---|---|---|---|
| Product Backlog | Metric | H | NSGA-II | SPEA2 | MOCell | SPEA2 | MOCell | MOCell |
| Local Government Project | HV | 2 | 2.87E − 11 | 2.87E − 11 | 8.68E − 01 | 2.44E − 03 | 2.87E − 11 | 2.87E − 11 |
| | | 3 | 6.87E − 08 | 5.45E − 09 | 3.85E − 08 | 4.43E − 04 | 1.89E − 08 | 9.76E − 06 |
| | | 4 | 3.88E − 04 | 8.01E − 06 | 5.87E − 09 | 8.08E − 01 | 6.11E − 06 | 8.86E − 04 |
| | | 5 | 2.84E − 05 | 5.43E − 10 | 2.87E − 11 | 2.83E − 03 | 3.50E − 08 | 5.62E − 07 |
| | IGD+ | 2 | 2.87E − 11 | 3.96E − 09 | 2.87E − 11 | 9.62E − 03 | 7.14E − 08 | 5.70E − 01 |
| | | 3 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 2.98E − 06 | 3.88E − 11 | 1.43E − 10 |
| | | 4 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 4.09E − 05 | 9.44E − 11 | 2.08E − 06 |
| | | 5 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 2.47E − 07 | 2.26E − 10 | 1.11E − 07 |
| Release Planner | HV | 2 | 3.96E − 07 | 2.87E − 11 | 5.32E − 10 | 9.06E − 01 | 4.79E − 05 | 7.44E − 09 |
| | | 3 | 5.27E − 06 | 9.44E − 11 | 1.38E − 05 | 3.50E − 08 | 2.25E − 01 | 1.47E − 01 |
| | | 4 | 7.10E − 04 | 8.12E − 09 | 2.14E − 01 | 7.43E − 05 | 2.19E − 02 | 3.66E − 07 |
| | | 5 | 4.22E − 05 | 4.27E − 06 | 3.88E − 11 | 7.01E − 01 | 4.44E − 02 | 1.09E − 03 |
| | IGD+ | 2 | 3.31E − 10 | 6.81E − 09 | 2.87E − 11 | 1.37E − 08 | 3.51E − 11 | 1.65E − 01 |
| | | 3 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 1.63E − 08 | 3.06E − 09 | 1.63E − 04 |
| | | 4 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 4.49E − 08 | 2.74E − 10 | 8.71E − 01 |
| | | 5 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 4.99E − 07 | 5.32E − 10 | 5.22E − 09 |
| Word Processor | HV | 2 | 1.02E − 09 | 2.87E − 11 | 2.87E − 11 | 8.14E − 03 | 1.47E − 02 | 1.53E − 02 |
| | | 3 | 3.64E − 10 | 2.87E − 11 | 2.74E − 10 | 7.10E − 04 | 5.32E − 10 | 2.98E − 06 |
| | | 4 | 6.07E − 06 | 1.77E − 09 | 9.31E − 10 | 8.63E − 02 | 3.21E − 08 | 2.40E − 06 |
| | | 5 | 2.87E − 01 | 4.34E − 04 | 1.37E − 04 | 1.17E − 01 | 5.79E − 05 | 2.87E − 02 |
| | IGD+ | 2 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 2.13E − 09 | 2.87E − 11 | 1.12E − 09 |
| | | 3 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 6.81E − 09 | 2.87E − 11 | 1.54E − 10 |
| | | 4 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 2.71E − 08 | 5.77E − 11 | 7.04E − 10 |
| | | 5 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 3.31E − 10 | 3.51E − 11 | 5.10E − 02 |
| RALIC | HV | 2 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 7.13E − 02 | 4.27E − 06 | 2.82E − 03 |
| | | 3 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 1.80E − 07 | 1.94E − 09 | 4.69E − 01 |
| | | 4 | 7.73E − 10 | 2.87E − 11 | 2.87E − 11 | 1.15E − 06 | 2.49E − 10 | 3.70E − 06 |
| | | 5 | 5.32E − 10 | 2.87E − 11 | 2.87E − 11 | 1.02E − 09 | 1.94E − 02 | 5.32E − 10 |
| | IGD+ | 2 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 1.35E − 09 | 6.37E − 11 | 9.27E − 03 |
| | | 3 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 2.49E − 10 | 1.54E − 10 | 7.44E − 09 |
| | | 4 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 1.62E − 09 | 6.37E − 11 | 5.65E − 02 |
| | | 5 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 1.62E − 09 | 7.03E − 11 | 9.44E − 08 |
| Synthetic-30 | HV | 2 | 1.17E − 01 | 3.88E − 04 | 7.78E − 03 | 5.35E − 01 | 2.76E − 02 | 7.36E − 02 |
| | | 3 | 5.32E − 10 | 1.12E − 09 | 2.87E − 11 | 2.14E − 01 | 1.53E − 02 | 8.48E − 01 |
| | | 4 | 6.51E − 06 | 2.87E − 11 | 2.87E − 11 | 1.84E − 04 | 1.47E − 01 | 1.14E − 01 |
| | | 5 | 6.04E − 02 | 2.82E − 03 | 2.19E − 04 | 3.83E − 01 | 9.19E − 02 | 3.59E − 01 |
| | IGD+ | 2 | 4.28E − 07 | 2.26E − 10 | 2.87E − 11 | 3.21E − 02 | 3.31E − 10 | 1.95E − 07 |
| | | 3 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 1.95E − 07 | 5.43E − 05 | 3.26E − 05 |
| | | 4 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 7.73E − 10 | 1.77E − 08 | 1.05E − 05 |
| | | 5 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 5.84E − 10 | 1.49E − 08 | 6.98E − 05 |
| Synthetic-50 | HV | 2 | 5.77E − 11 | 5.23E − 11 | 7.49E − 04 | 8.70E − 08 | 1.10E − 02 | 1.77E − 08 |
| | | 3 | 7.44E − 09 | 1.37E − 08 | 9.31E − 10 | 2.49E − 01 | 1.47E − 02 | 5.10E − 02 |
| | | 4 | 6.16E − 05 | 1.48E − 09 | 2.76E − 04 | 7.34E − 01 | 5.70E − 03 | 3.45E − 06 |
| | | 5 | 1.62E − 09 | 2.87E − 11 | 2.87E − 11 | 1.60E − 01 | 3.09E − 02 | 6.36E − 01 |
| | IGD+ | 2 | 9.64E − 01 | 4.40E − 10 | 3.18E − 11 | 3.66E − 09 | 2.26E − 10 | 9.44E − 08 |
| | | 3 | 1.11E − 07 | 2.87E − 11 | 2.49E − 10 | 3.31E − 10 | 2.68E − 07 | 9.41E − 01 |
| | | 4 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 1.02E − 09 | 3.88E − 11 | 1.73E − 02 |
| | | 5 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 2.74E − 10 | 1.04E − 10 | 6.04E − 02 |
| Synthetic-100 | HV | 2 | 3.71E − 05 | 3.71E − 05 | 1.20E − 07 | 9.99E − 01 | 6.56E − 05 | 6.56E − 05 |
| | | 3 | 6.05E − 01 | 6.04E − 01 | 1.02E − 07 | 9.99E − 01 | 3.33E − 02 | 3.33E − 02 |
| | | 4 | 3.51E − 11 | 3.51E − 11 | 3.88E − 11 | 9.99E − 01 | 9.76E − 01 | 9.76E − 01 |
| | | 5 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 9.99E − 01 | 6.73E − 04 | 6.73E − 04 |
| | IGD+ | 2 | 1.31E − 07 | 1.31E − 07 | 2.26E − 10 | 9.99E − 01 | 3.09E − 04 | 3.09E − 04 |
| | | 3 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 9.99E − 01 | 1.60E − 01 | 1.60E − 01 |
| | | 4 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 9.99E − 01 | 4.58E − 04 | 4.58E − 04 |
| | | 5 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 9.99E − 01 | 9.41E − 01 | 9.41E − 01 |
| Synthetic-200 | HV | 2 | 8.12E − 09 | 7.79E − 03 | 2.87E − 11 | 3.91E − 01 | 2.37E − 02 | 2.14E − 01 |
| | | 3 | 5.71E − 09 | 1.73E − 02 | 2.87E − 11 | 8.01E − 06 | 2.68E − 05 | 2.07E − 04 |
| | | 4 | 4.49E − 08 | 4.79E − 05 | 2.87E − 11 | 6.28E − 07 | 1.20E − 07 | 4.58E − 04 |
| | | 5 | 1.54E − 10 | 2.87E − 11 | 2.87E − 11 | 7.34E − 01 | 2.56E − 02 | 4.10E − 04 |
| | IGD+ | 2 | 1.53E − 02 | 2.87E − 11 | 4.73E − 09 | 1.54E − 10 | 3.50E − 08 | 2.04E − 01 |
| | | 3 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 2.74E − 10 | 1.02E − 07 | 3.96E − 05 |
| | | 4 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 7.76E − 11 | 1.63E − 08 | 2.10E − 08 |
| | | 5 | 2.87E − 11 | 2.87E − 11 | 2.87E − 11 | 3.17E − 11 | 1.23E − 09 | 5.22E − 09 |
| **Overall** | HV | | 6.17E − 98 | 2.84E − 139 | 1.09E − 110 | 2.71E − 11 | 3.29E − 06 | 3.40E − 01 |
| | IGD+ | | 9.03E − 110 | 1.74E − 243 | 5.99E − 246 | 2.30E − 98 | 6.82E − 105 | 9.85E − 01 |

## C ADDITIONAL DATA

### C.1 Statistical Significance

For the experiments comparing BEARS to other release planning methods (Section 5 and Appendix A), we have checked that the observed differences in hypervolumes between BEARS and the other methods are statistically significant using Mann-Whitney U test with $p < .05$. All p-values are reported in Table 9.

### C.2 Illustrative Examples of Generated Shortlists

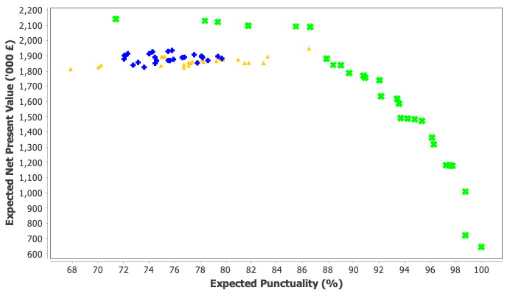
To understand the practical significance of the differences between BEARS and the other methods in our first and second experiments, it is useful to look at a few examples of shortlists generated by the methods on specific release planning problems. Figures 6 and 7 show examples of shortlists generated by all release planning methods for each product backlog and a planning horizon $H = 3$. These examples all correspond to the first of the 30 runs of each method. For each release planning problem, the figure shows how much the shortlist generated by BEARS dominates the shortlists generated by the other methods.

In Figure 6, we observe that the dominance of BEARS over methods that ignore uncertainty is important in practice. For each of these examples, we can make similar observations to those made about the illustrative example in the paper (Section 5.1.5).

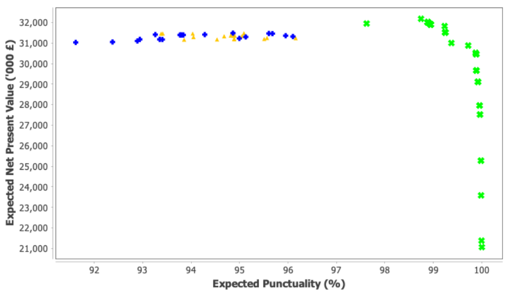
In Figure 7, we see that the difference between BEARS and methods that analyse uncertainty assuming fixed-scope releases is smaller but still important enough to justify using BEARS for projects with fixed-date release cycles.

### C.3 Run-times

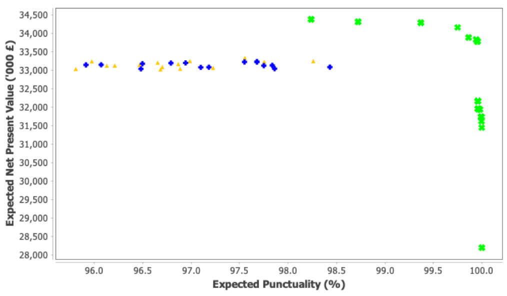
Table 10 shows the average run-time of each method for each release planning problem.



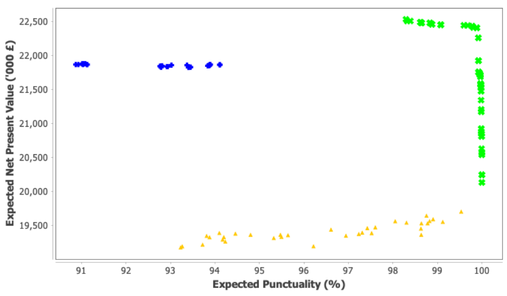
(a) Local Government



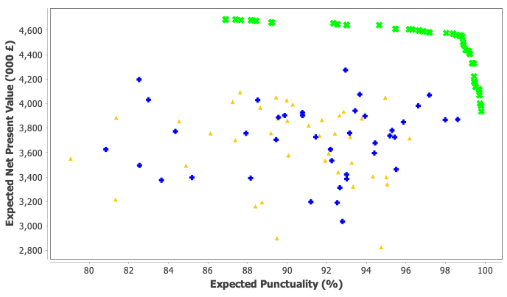
(b) Release Planner



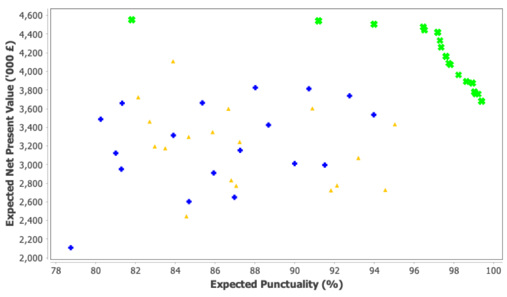
(c) Word Processor



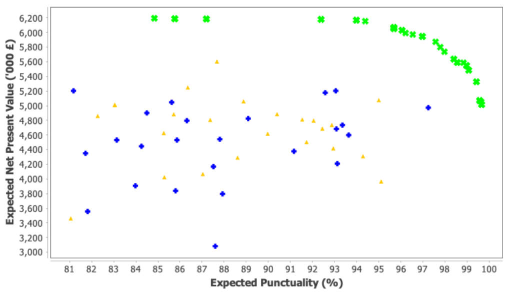
(d) RALIC



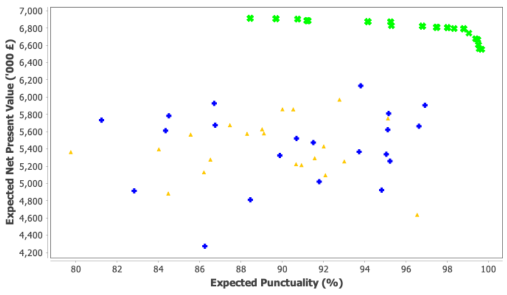
(e) Synthetic-30



(f) Synthetic-50

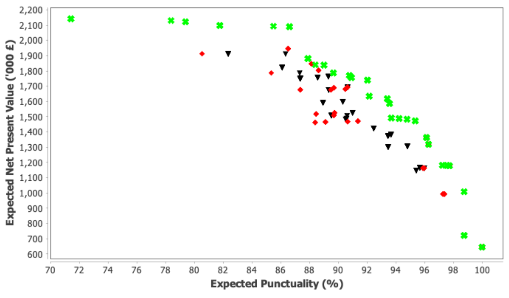


(g) Synthetic-30

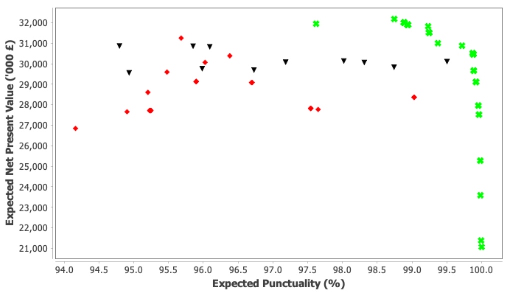


(h) Synthetic-50

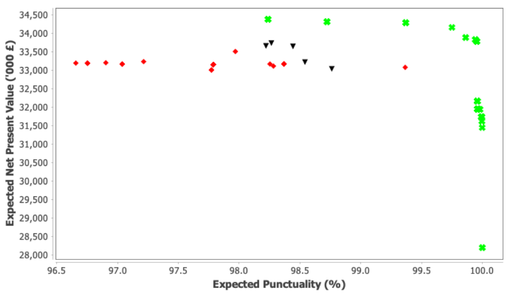
Fig. 6. Examples of shortlists generated by BEARS (green crosses), NPV-*deterministic* (yellow triangles) and VP-*deterministic* (blue diamonds). These examples correspond to the first of 30 runs of each method.



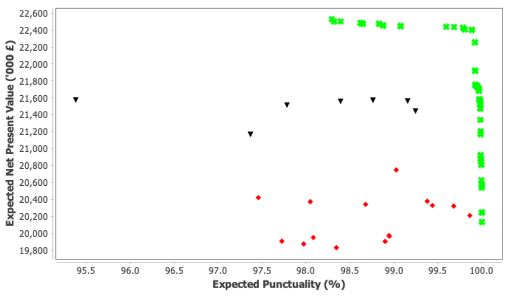
(a) Local Government



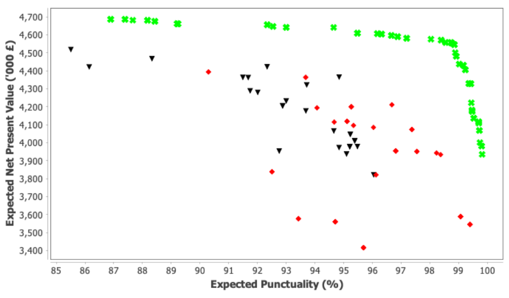
(b) Release Planner



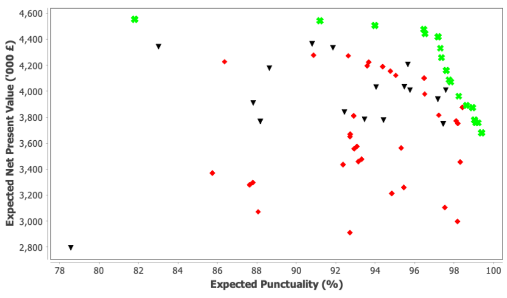
(c) Word Processor



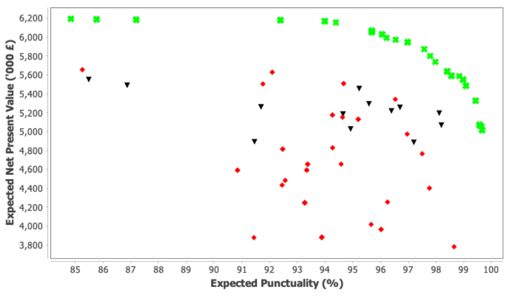
(d) RALIC



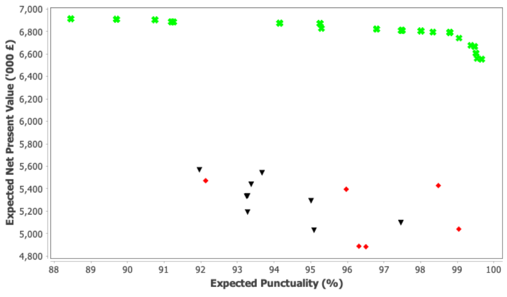
(e) Synthetic-30



(f) Synthetic-50



(g) Synthetic-30



(h) Synthetic-50

Fig. 7. Examples of shortlists generated by BEARS (green crosses), NPV-*fixed-scope* (black triangles) and VP-*fixed-scope* (red diamonds). These examples correspond to the first of 30 runs of each method.

Table 9. Statistical Significance (p-value) of the observed difference in hypervolume between BEARS and other methods over 30 runs using Mann-Whitney U test.

| | | BEARS *vs.* | | | |
|---|---|---|---|---|---|
| Product Backlog | H | VP-*deterministic* | NPV-*deterministic* | VP-*fixed-scope* | NPV-*fixed-scope* |
| Local Government Project | 2 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $5.77E-11$ |
| | 3 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 4 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 5 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| Release Planner | 2 | $2.11E-07$ | $2.10E-08$ | $4.58E-04$ | $1.25E-02$ |
| | 3 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $3.51E-11$ |
| | 4 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 5 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| Word Processor | 2 | $3.01E-01$ | $6.04E-02$ | $4.87E-01$ | $3.85E-02$ |
| | 3 | $2.87E-11$ | $6.37E-11$ | $1.15E-10$ | $3.06E-09$ |
| | 4 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 5 | $2.87E-11$ | $2.74E-10$ | $5.84E-10$ | $2.87E-11$ |
| RALIC | 2 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 3 | $6.37E-11$ | $2.87E-11$ | $3.31E-10$ | $2.05E-10$ |
| | 4 | $3.18E-11$ | $2.87E-11$ | $3.18E-11$ | $2.87E-11$ |
| | 5 | $3.51E-11$ | $2.87E-11$ | $9.44E-11$ | $3.51E-11$ |
| Synthetic-30 | 2 | $2.87E-11$ | $2.87E-11$ | $4.73E-11$ | $6.41E-10$ |
| | 3 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 4 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 5 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| Synthetic-50 | 2 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $7.39E-08$ |
| | 3 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 4 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 5 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| Synthetic-100 | 2 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 3 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 4 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 5 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| Synthetic-200 | 2 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 3 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 4 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| | 5 | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ | $2.87E-11$ |
| **Overall** | | $5.43E-116$ | $3.21E-149$ | $1.36E-104$ | $2.71E-114$ |

Table 10. Release planning methods' run-times in seconds

| Product Backlog | H | VP-*deterministic* | NPV-*deterministic* | VP-*fixed-scope* | NPV-*fixed-scope* | BEARS |
|---|---|---|---|---|---|---|
| Local Gov. Project | 2 | 2 | 3 | 10 | 18 | 45 |
| | 3 | 2 | 4 | 12 | 20 | 48 |
| | 4 | 3 | 4 | 15 | 21 | 60 |
| | 5 | 3 | 5 | 17 | 25 | 65 |
| Release Planner | 2 | 2 | 3 | 10 | 15 | 37 |
| | 3 | 2 | 4 | 14 | 25 | 55 |
| | 4 | 3 | 5 | 16 | 30 | 70 |
| | 5 | 5 | 8 | 20 | 35 | 95 |
| Word Processor | 2 | 2 | 4 | 18 | 36 | 95 |
| | 3 | 3 | 7 | 20 | 47 | 135 |
| | 4 | 4 | 8 | 23 | 58 | 155 |
| | 5 | 6 | 10 | 26 | 65 | 163 |
| RALIC | 2 | 4 | 10 | 37 | 85 | 153 |
| | 3 | 5 | 12 | 45 | 102 | 192 |
| | 4 | 5 | 12 | 54 | 123 | 225 |
| | 5 | 7 | 15 | 68 | 150 | 259 |
| Synthetic-30 | 2 | 2 | 6 | 15 | 33 | 75 |
| | 3 | 3 | 7 | 17 | 42 | 95 |
| | 4 | 3 | 9 | 17 | 40 | 112 |
| | 5 | 5 | 11 | 20 | 49 | 125 |
| Synthetic-50 | 2 | 5 | 9 | 20 | 68 | 158 |
| | 3 | 7 | 11 | 25 | 75 | 170 |
| | 4 | 7 | 12 | 29 | 84 | 188 |
| | 5 | 9 | 14 | 37 | 96 | 205 |
| Synthetic-100 | 2 | 6 | 13 | 56 | 75 | 229 |
| | 3 | 6 | 16 | 75 | 94 | 284 |
| | 4 | 7 | 20 | 85 | 110 | 302 |
| | 5 | 7 | 24 | 97 | 115 | 357 |
| Synthetic-200 | 2 | 9 | 20 | 83 | 112 | 288 |
| | 3 | 9 | 21 | 85 | 122 | 365 |
| | 4 | 10 | 22 | 95 | 130 | 486 |
| | 5 | 10 | 30 | 120 | 150 | 533 |